Keynote Paper

Elastic Circuits

Josep Carmona, Jordi Cortadella, *Member, IEEE*, Mike Kishinevsky, *Senior Member, IEEE*, and Alexander Taubin, *Senior Member, IEEE*

***Abstract*—Elasticity in circuits and systems provides tolerance to variations in computation and communication delays. This paper presents a comprehensive overview of elastic circuits for those designers who are mainly familiar with synchronous design. Elasticity can be implemented both synchronously and asynchronously, although it was traditionally more often associated with asynchronous circuits. This paper shows that synchronous and asynchronous elastic circuits can be designed, analyzed, and optimized using similar techniques. Thus, choices between synchronous and asynchronous implementations are localized and deferred until late in the design process.**

***Index Terms*—Asynchronous circuits, design, elastic circuits, electronic design automation, latency-insensitive design, latency-tolerant design, variability.**

I. Introduction

VARIABILITY is one of the main impediments for successful circuit design. Uncertainties in estimating delays force designers to include conservative margins that prevent circuits from working at the performance potentially achievable for a specific technology.

Asynchronous circuits [105] have often been proposed as a solution to the challenges of variability. However, many designers still consider asynchronous circuits as an esoteric class of devices with questionable value. The lack of nondisruptive electronic-design-automation (EDA) flows has been one of the main reasons why designers have been reluctant to adopt this paradigm.

Recently, synchronous solutions to delay variability have been proposed. Latency insensitivity [22] was introduced as a method to design circuits tolerant of the latency variability of computations and communications. From a broader standpoint, latency-insensitive design can be considered as a discretization of asynchronous design.

Manuscript received June 17, 2008; revised February 4, 2009, and, June 22, 2009. Current version published September 18, 2009. This work was supported in part by the Comision Interministerial de Ciencia y Tecnologia under Grant TIN2007-66523, by the Elastix Corporation, and by Intel Corporation. This paper was recommended by Associate Editor S. Nowick.

J. Carmona is with the Department of Software, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain.

J. Cortadella is with the Department of Software, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain, and also with Elastix Solutions S.L., 08022 Barcelona, Spain.

M. Kishinevsky is with Intel Corporation, Hillsboro, OR 97124 USA.

A. Taubin is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2030436

This paper presents a review of elastic circuits. In this context, *elasticity* refers to the property of a circuit to adapt its activity to the timing requirements of its computations, communications, and operating conditions. Elasticity does not make any assumption about the specific implementation of the circuit.

This paper also presents a unified view of elastic circuits and shows that they can be designed and analyzed with common and conventional design flows. The decision as to whether to select a synchronous or asynchronous implementation can be postponed until the latest stages of the synthesis flow.

A. Why Elastic Circuits?

The scaling down to sub-100-nm technologies is having an important impact on power consumption, variability, and circuit complexity. Designing circuits with rigid clocks imposes rigorous timing constraints and prevents the circuit from adapting its functionality to dynamically changing operating conditions.

The time required to transmit data between two different locations is governed by the distance between them and often cannot be accurately known until the chip layout. Traditional design approaches require fixing the communication delays up front, and these are difficult to amend when layout information finally becomes available.

Elasticity is a paradigm that provides the possibility of tolerating timing variations in the computations and communications of a circuit and its environment. In addition, the modularity of elastic systems promises novel methods for microarchitectural design that can use variable-latency components and tolerate static and dynamic changes in communication latencies.

This tolerance may enable schemes to adapt the voltage and speed of a circuit to the operating conditions. For example, in an elastic system, a component may automatically reduce its speed when the temperature becomes too high without affecting the correctness of the interaction with the rest of components.

More formally, elasticity removes timing constraints from the nonfunctional requirements of a system. The more timing constraints are removed, the more elasticity is acquired by the system. These properties are often associated with ease of design, e.g., simpler timing closure.

B. Cost of Elasticity

Elasticity is also associated with an extra cost that may become prohibitive in some cases. Fig. 1 graphically shows the tradeoff between elasticity and area overhead for different

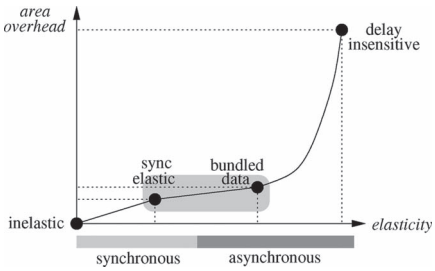

Fig. 1. Overhead associated with different types of elasticity.

classes of circuits. The simplest class is referred to as *synchronous elasticity* [22], [31]. For a small control overhead, the circuit can acquire a certain degree of cycle-level elasticity that can make it tolerant of latency uncertainties.

Elasticity can reach finer levels of granularity by making the circuit asynchronous. The cheapest style is the class of *bundled-data* circuits [105]. In this paper, we show how synchronous-elastic and bundled-data circuits can be implemented with the same datapath. They only differ in the small control to generate the clocking. Finally, there is a large variety of asynchronous circuit styles, with increasing elasticity and overhead.

The most elastic and robust class is the one called delay-insensitive (DI) circuits [78]. In their purest form, DI circuits are tolerant of any delay variability of their components (gates and wires). Unfortunately, the area overhead is extremely large, and the set of DI circuits with practical interest is rather small. For this reason, relaxed versions of DI circuits are often used instead.

This paper will focus on those classes of circuits that can provide elasticity with small overhead. These are the circuits in the shadowed area shown in Fig. 1: synchronous elastic circuits and asynchronous circuits with bundled data. Some aspects of the other classes of asynchronous circuits will also be discussed.

C. Essential Role of EDA

EDA tools have been essential for the rapid growth of electronics. Nowadays, it is inconceivable to face the design of a complex system without resorting to a complete EDA design flow that covers all the phases from the specification down to the physical implementation.

Static timing analysis [98] (STA) is one of the fundamental phases in circuit design. The goal of STA is to estimate the delay of a circuit and detect possible timing errors. Synchronous circuits include flip-flops and latches that enforce the synchronization of multiple paths. This synchronization provides a set of cut points that simplify STA considerably, since the analysis is reduced to combinational paths between cut points.

Traditionally, one of the main reasons for the reluctance of designers to adopt asynchronous circuits has been the lack of EDA flows with similar maturity to those for synchronous circuits, and STA tools have been the main target of criticism. The unpredictability in the timing of events makes it difficult to adopt existing STA tools for the analysis of asynchronous circuits.

This paper will show how elasticity can be handled by a common EDA flow in which the datapath can be designed independently from the clocking scheme, either synchronous or asynchronous. Synthesis, analysis, and optimization can be supported by similar tools and methods, and only the specific details related to the generation of the clock signals must be treated differently.

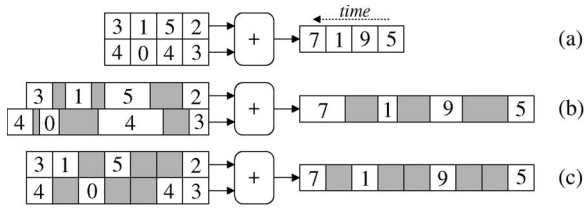

Fig. 2. Different types of elasticity. (a) Inelasticity. (b) Asynchronous elasticity. (c) Synchronous elasticity. Shaded boxes denote idle periods.

D. Organization of this Paper

Section II introduces the concept of elasticity with synchronous and asynchronous variations. Section III discusses common implementability aspects of elasticity that are independent of the synchronous or asynchronous nature of the control. Sections IV and V present particular aspects of the implementation of asynchronous and synchronous protocols and latch controllers, respectively. Section VI proposes a common EDA flow for elastic circuits. Completion detection is one of the key aspects that determines the performance of an elastic circuit. Different schemes are discussed in Section VII. Performance analysis and different techniques for performance optimization will be discussed in Section VIII. A final discussion on the subject and some examples are presented in Section IX.

II. Elasticity

Elasticity is a concept that has been used in various contexts to refer to the flexibility of a system to adapt to the variability of delays. The delay uncertainty can be either produced by the system (e.g., voltage and temperature variations) or by the environment (e.g., unpredictable input data rates).

Fig. 2 shows an attempt to graphically describe the meaning of elasticity. The system is a simple adder that receives two input data and produces their sum. The behavior of a conventional inelastic system is shown in Fig. 2(a), where the adder expects data and produces the result at each cycle. Each box in the diagram represents a cycle, and the contents of the box represent the value at the input or output of the adder. In an inelastic system, the behavioral correctness depends on a global timing synchronization that cannot be dynamically modified.

Fig. 2(b) shows the behavior of an elastic system. The incoming data at each *elastic channel* may arrive at different time instants, while the adder may also take a variable delay to produce the result. Since there is no global synchronization, a handshake protocol is required to synchronize the sender and the receiver at each elastic channel. In an elastic system, the behavioral correctness does not depend on the absolute timing occurrence of the events. When delay information is available, the handshake protocol might be simplified whenever explicit synchronization is not required.

Fig. 2(c) shows a restricted class of elasticity. In this case, there is a global clock that synchronizes the events. However, the events may occur at arbitrary cycles, i.e., there may be *idle* cycles at each channel. This is a type of *discrete* elasticity still

synchronized with a global clock signal that we will refer as *synchronous elasticity*, in contrast to the more generic type of elasticity that we will often refer to as *asynchronous elasticity*.

A. Formalizing Elasticity

Different forms of elasticity have been studied in the last decade, thus providing various formal models with strong similarities and subtle differences. All these models require a notion of equivalence that defines the conditions under which two systems can be considered to have the same behavior. They also characterize the valid transformations to incorporate elasticity into a system.

The formal models for elasticity are reminiscent to the notion of *observational equivalence* [77], in which two systems are equivalent when an external agent cannot differentiate them by looking at their observable traces. In case of elasticity, the observable traces would be the ones obtained after hiding the *idle* cycles or events, denoted by τ in [77].

In the synchronous domain, the theory of *patient processes* [22] introduced the notion of *latency equivalence*, extended by the theory of *elastic networks* in [61]. In the asynchronous domain, the concepts of *slack elasticity* [70] and *flow equivalence* [46] denote similar equivalences, adapted to asynchronous nature of timing (we refer the reader to those papers for further details on these formal models).

B. Asynchronous Elasticity

Elasticity, in its broader sense, has been formalized in different ways and contexts. Probably, the earliest and closest term approaching elasticity was proposed by Molnar *et al.* [78] in the Macromodules project, introducing the concept of a *DI module*.

A DI module M interacts with its environment through input and output signals. Between the module and its environment, there are arbitrary interface delays that defer the observation of the inputs in the module and outputs in the environment. These delays are represented by a *foam–rubber wrapper* surrounding the module.

Strictly, a DI circuit is one that behaves correctly regardless of the delays of its components (gates and wires). Due to this stringent constraint, the class of DI circuits is rather small [71].

A practical relaxation of this constraint was proposed by Seitz [100], introducing the concept of *self-timed systems*. In this context, the modules of the system are assumed to work with local timing constraints, e.g., negligible wire delays, thus relegating delay insensitivity to the interface of the modules only. Since then, a large body of knowledge has been created on the design and formalization of asynchronous circuits [9], [18], [38], [57], [73], [82], [105].

C. Synchronous Elasticity

When using a clock to discretize elasticity, the delay insensitivity is restricted to multiples of clock cycles. This direction aims at exploring a tradeoff between the flexibility of asynchronous systems and the extensive computer-aided-design tool support for synchronous design.

The first formal approach to synchronous elasticity was proposed by Carloni *et al.* [22], coining the term *latency insensitivity*. After that, several authors have proposed different schemes for synchronous elasticity [31], [52], [92].

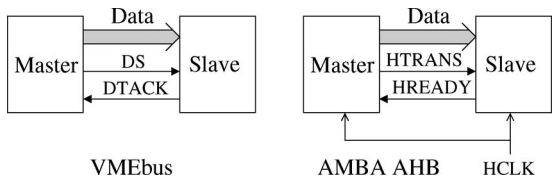

Fig. 3. Handshake signals for synchronous and asynchronous elastic buses.

The behavior observed in these approaches corresponds to the one shown in Fig. 2(c). The handshake signals may change at each cycle, and their value determines the validity of the data being transferred during the cycle. The most valuable advantage of synchronous elasticity is the capability of designing circuits with conventional design flows using STA.

D. Examples of Elasticity

There are already some examples of elasticity that are well known by circuit designers. Elasticity is typically useful in those designs where components with different speeds must interact. This is the reason why the most popular buses use elastic protocols, e.g., the VMEbus [51], AMBA AHB [3], or OCP [88]. In all these cases, elasticity requires a pair of signals to implement a handshake protocol. Single-track protocols are also possible, requiring only one wire [11], [110].

Fig. 3 shows a simplified scheme of the handshake signals for the VME (asynchronous) and AMBA AHB (synchronous) buses. In both cases, there is a signal going from master to slave to indicate the availability of information in the bus. Similarly, there is another signal from slave to master to indicate the completion of the transfer. In OCP, a similar synchronous handshaking protocol is performed with the signals called master command (MCmd) and slave command accept (SCmdAccept).

In asynchronous protocols, data transfers are indicated by the events of the handshake signals, e.g., data strobe (DS) and data-transfer acknowledge (DTACK) in the VMEbus. In synchronous protocols, the initiation and completion of transfers are indicated by the value of the handshake signals at the clock edges. In the AMBA AHB bus, the HTRANS signal (two bits) may indicate an IDLE cycle when no transfer must be performed. The slave can also indicate that it has not been able to accept the transfer (HREADY $= 0$), thus forcing the master to maintain the same data on the subsequent cycles until the transfer has been completed (HREADY $= 1$).

III. Implementing Elasticity

Elasticity is inherently associated with the concept of *distributed control*. The communication between two elastic components requires bidirectional information to manage the flow of data correctly. In the *forward* direction, the validity of data must be communicated from the sender to the receiver. In the *backward* direction, the availability of resources to consume data must be communicated from the receiver to the sender.

Different nomenclatures have been used to name the control signals for elastic communication. In asynchronous design, the most popular names are *request* and *acknowledge* for the forward and backward signals, respectively. In synchronous design, *valid* and *stop* (or *stall*) are often used.

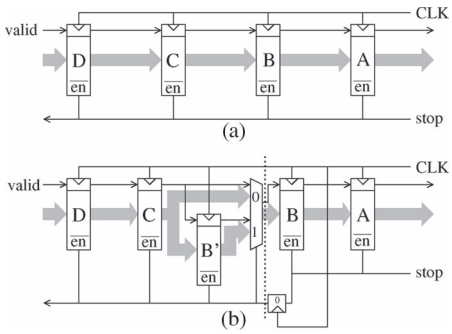

Fig. 4. Distributed control for elasticity.

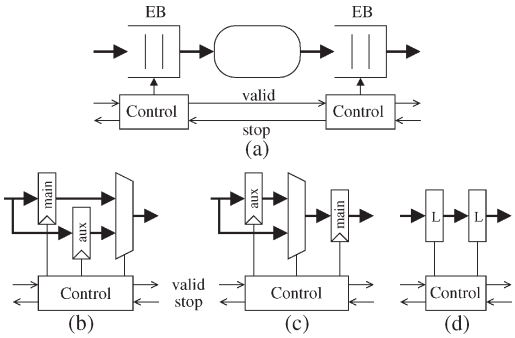

Fig. 5. Implementation of EBs.

An instructive explanation of the requirements for elastic communication can be found in [52]. We briefly sketch these requirements in the example shown in Fig. 4. The synchronous pipeline contains four edge-triggered registers with an enable control signal ($\overline{en}$ = not enabled). Fig. 4(a) shows a scheme in which the *valid* signal is distributed along the stages, whereas the *stop* signal is global. In this scheme, all the stages are stalled when the environment asserts the *stop* signal.

Unfortunately, a global *stop* signal may be unacceptable for a large system with long wire delays. However, a distributed control requires a more complicated scheme to handle the *back-pressure* generated by the assertion of the *stop* signal. Fig. 4(b) shows a possible implementation of the distributed *stop*, assuming that a cycle is taken to propagate from registers A and B to registers C and D. When *stop* is asserted, the values stored at A and B must be kept for the next cycle. However, a new value is simultaneously *flying* from C to B and another from D to C.

In the previous scenario, extra storage is necessary to accommodate the new incoming data from C to B while preserving the previous value of B. This is the purpose of the auxiliary register B' in Fig. 4(b). When the stop signal is deasserted, the value stored at B' will be first delivered to B before the new incoming data from C is transferred.

In general, the communication between two elastic modules, either synchronous or asynchronous, requires extra storage to capture the incoming messages that cannot be delivered because of the back-pressure. Fig. 5(a) shows a general scheme in which each channel has an elastic buffer[1] (EB) to store the incoming values.

[1]Also called *relay station* in [22]. Note that while this paper describes EBs capable of holding two data items, in general, EBs are elastic first-in-first-outs (FIFOs) of arbitrary finite capacity.

The capacity of an EB defines the maximal number of data items that can be simultaneously stored inside the buffer. The correct operation of elastic circuits requires minimal constraints on the size of EBs. In synchronous elasticity, the capacity of an EB C is a natural number that must satisfy the following constraint: $C \geq L_f + L_b$, where L_f is the forward-propagation latency of the data and valid signals and L_b is the backward-propagation latency of the stop signal measured in the number of clock cycles. In the usual case of an EB with equal forward and backward latencies of one clock cycle $L_f = L_b = 1$, it is sufficient to have EBs capable of storing two data items. One location is used for propagation of data item in absence of back-pressure (as in a regular register), while the other location stores the second data item which is sent by the sender before it could see the blocking stop request. The proof of the earlier fact has been done in the context of relay stations in [21] and [68] and follows from the analysis of marked-graph (MG) models corresponding to the elastic designs. Such models will be described in Section IV-B.

Fig. 5(b) and (c) shows two possible implementations of an EB using edge-triggered registers. Other implementations are also possible, but in all of them, a multiplexor is required to select data from the main flow or from the auxiliary storage. As will be discussed as follows in the next section, a more efficient implementation using transparent latches is possible; however, the use of flip-flop-based registers can be more convenient for timing analysis or for the implementation in field-programmable gate arrays that do not provide support for latches.

A. Latch-Based Implementations of EBs

Efficient implementations of EBs can be derived using level-sensitive latches, either in asynchronous designs [34], [59] or in synchronous designs [31], [52]. These implementations are based on the observation that each edge-triggered flip-flop is composed of two level-sensitive latches (master and slave) [27] that can potentially store different data if they can be controlled independently. The resulting approach is shown in Fig. 5(d) and is more efficient in area, delay, and power as compared to the flip-flop-based implementations.

Using the latch-based implementation of EBs shown in Fig. 5(d), we can compare three schemes with different degrees of elasticity. Fig. 6(a) shows a conventional inelastic circuit with a global clock connected to all flip-flops through a clock tree. Fig. 6(b) shows a synchronous elastic scheme in which the *valid/stop* signals are distributed along multiple stages. A global clock remains to synchronize the control signals. Finally, Fig. 6(c) shows a totally distributed (asynchronous) scheme in which only locally generated clock signals are used to synchronize adjacent stages. The removal of the global clock requires the addition of local delays (shown in rounded boxes) to match the corresponding delays in the datapath.

It is important to realize that the schemes in Fig. 6(b) and (c) are very similar. The datapaths and the locally generated enabling signals are identical. The only differences are the logic and timing to generate the enabling pulses for the master/slave latches. These schemes can also be generalized to incorporate logic between the master and slave latches.

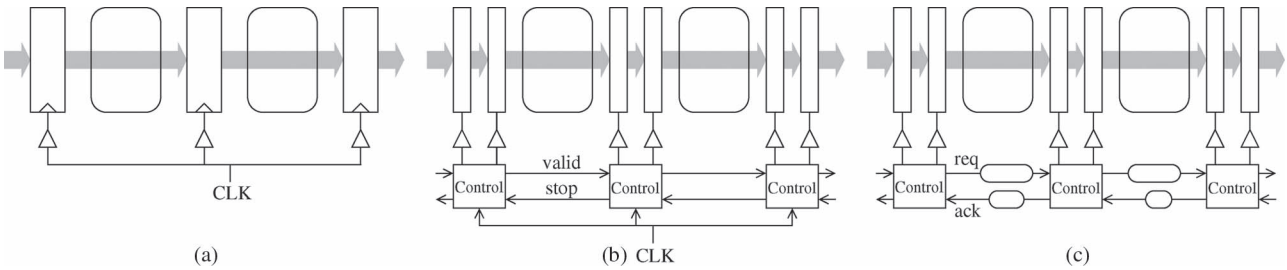

Fig. 6. (a) Synchronous inelastic circuit. (b) Synchronous elastic circuit. (c) Asynchronous elastic circuit (with matched delays).

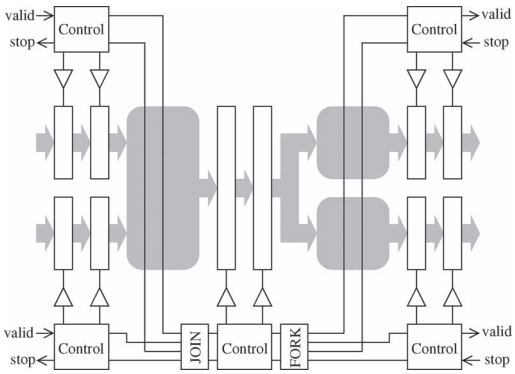

Fig. 7. Synchronous elastic module with multiple inputs and outputs.

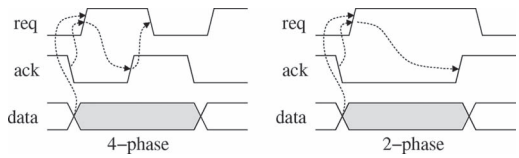

Fig. 8. Four- and two-phase protocols for data communication.

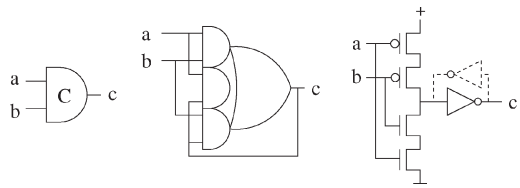

Fig. 9. Muller's C-element with two different implementations.

B. Extension to Generic Elastic Netlists

The designs shown in Fig. 6 can be easily extended to any arbitrary netlist in which the elastic modules have multiple inputs and outputs connected to other elastic modules.

An example of such extension is shown in Fig. 7. The controller in the middle of the figure is synchronized with the neighboring controllers. The Join block combines the handshake signals of the input modules with the ones of the controller. Similarly, the Fork block combines the handshake signals for the output modules. Intuitively, the Join block implements the conjunction of *valid* signals, whereas the Fork block implements the disjunction of *stop* signals.[2]

The exact details on how these blocks can be implemented will be discussed in Sections IV and V.

C. Adding Bubbles

One of the properties of elastic circuits is that they accept the insertion of latches at arbitrary locations while preserving the behavior of the circuit. This can be done either for asynchronous circuits [70], [80] or synchronous circuits [21], [61]. To preserve the original behavior, these latches must be initialized with *empty* (nonvalid) data, often referred to as *bubbles*, thus maintaining the same amount of valid data after the insertion. In this way, the latches merely behave as delays in the datapath.

The main reason for the insertion of *bubbles* is performance: This insertion can compensate for the performance degradation produced by unbalanced fork–join structures of some designs. Empty EBs can also be inserted to pipeline long wires for the purpose of fixing timing-closure exceptions in synchronous elastic designs or removing performance bottlenecks in asynchronous elastic designs. Section VIII-B will discuss some strategies to improve the performance by adding bubbles in the circuit.

[2]The same scheme applies for an asynchronous circuit with *req/ack* handshake signals.

IV. Asynchronous Elasticity

This section reviews different approaches for designing asynchronous elastic circuits with synchronization schemes like the one shown in Fig. 6(c). The synchronization schemes are characterized by the protocols committed by the handshake signals (`req` and `ack`). We can classify the protocols into two main families: four-phase and two-phase. We will review two classical schemes for each family, which are interesting due to their elegance, and help to illustrate the important points of this section. Many other schemes that supercede the ones presented in this section have been presented in the last two decades and will be shortly highlighted at the end of the section.

A. Four-Phase Signaling

A four-phase protocol requires four events of the handshake signals for each communication (see Fig. 8).

$$\texttt{req}\uparrow \quad \texttt{ack}\uparrow \quad \texttt{req}\downarrow \quad \texttt{ack}\downarrow$$

The first two are called *signaling* events, whereas the last two are known as the *return-to-zero* events. One of the most popular protocols is based on Muller's C-element [79], shown in Fig. 9, that synchronizes the events at the inputs and produces an event at the output. The logic equation for a C-element is

$$c^{\text{next}} = ab + c(a + b)$$

where c^{next} represents the next state value of the output c. A C-element preserves the value of c when the inputs are different and propagates the value of the inputs when they are the same. This behavior can be extended to C-elements with

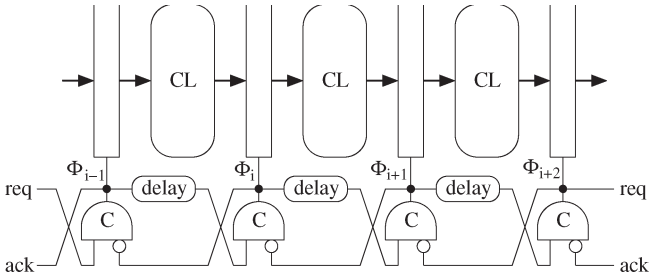

Fig. 10. Synchronization control based on Muller's pipeline.

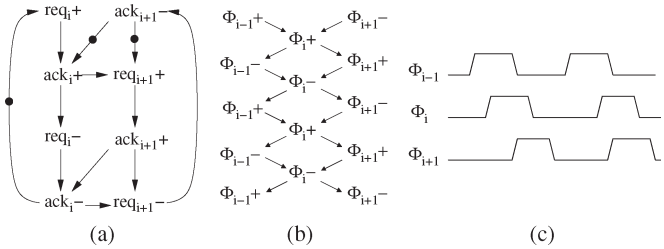

Fig. 11. Behavioral model of Muller's pipeline. (a) STG specification. (b) Enable signals of adjacent latches. (c) Timing diagram.

n inputs: When all n inputs are high, the output of the n input C-element can go high, and when all inputs are low, the output can go low. Two different implementations of the C-element are shown in Fig. 9.

Fig. 10 shows an asynchronous pipeline with a four-phase handshake control based on Muller's protocol. Every latch receives an enable signal (Φ) coming from a C-element of the control. For the enable signal to rise $(\Phi \uparrow)$, each C-element waits for the arrival of new data from the previous stage $(\texttt{req} \uparrow)$ and the completion of the communication of the next stage $(\texttt{ack} \downarrow)$. This models the situation in which data are available at the input (setup constraint) and the data at the output have already been consumed (hold constraint). After $\Phi \uparrow$, the control continues with the remaining events of the four-phase protocol.

The delay inserted between the Φ signal and the `req` signal of the next stage guarantees the setup constraint, i.e., that data are stable when $\texttt{req} \uparrow$ occurs.

A well-known property of circuits based on transparent latches and nonoverlapping clocking scheme is that no data transfers occur simultaneously between two adjacent stages. This property is the same as what it is observed in flip-flop-based synchronous circuits, in which data transfers occur sequentially from master to slave or from slave to master latches (that are hidden inside the master–slave implementation of the flip-flops), but not simultaneously. A more detailed discussion about performance in elastic circuits will be presented in Section VIII.

B. Behavioral Model of Muller's Pipeline

Fig. 11 shows a behavioral (untimed) model of Muller's pipeline, represented by a signal transition graph (STG [28], [97]). An STG is an interpreted Petri net [81] in which events (also called transitions) represent signal transitions.[3] A particular class of Petri nets, called MGs, are often used to represent the behavior of pipelines. The nodes (transitions) of an MG represent the events, and the arcs represent the causality relations between pairs of events, whereas the tokens (marks on the arcs) represent the state of the system. In an MG, an arc can either be marked with a token or unmarked. When all input arcs of an event are marked, an event can be performed (can be fired), removing the tokens from the incoming edges and placing one token on each outgoing edge. An MG has an initial marking, and the operation of the system can be described by the set of all possible sequences of markings that start with that initial marking (we refer the reader to [32], [81], [82] for deeper treatments of Petri nets, MGs, and STGs).

The STG in Fig. 11(a) shows the relationship between the `req`/`ack` signals of two adjacent stages of a pipeline. The important causality relations are the ones represented by the arcs

$$\texttt{req}_i+ \rightarrow \texttt{ack}_i + \qquad \text{(setup constraint)}$$
$$\texttt{req}_{i+1}- \rightarrow \texttt{ack}_{i+1} - \qquad \text{(hold constraint)}.$$

Fig. 11(b) shows the behavior of the pipeline referring only to the Φ signals of the latches. Finally, Fig. 11(c) shows the same behavior with a timing diagram. The previous causality relations must be accompanied with appropriate timing constraints to meet the timing requirements of the circuit. In particular, *time borrowing* is essential for efficient latch-based designs. This will be discussed in Section VI-B.

[3]For historical reasons, $+$ and $-$ are used to denote rising and falling signal transitions, respectively. For example, $ack+$ indicates a rising transition on signal ack, and $req-$ indicates a falling transition on signal req.

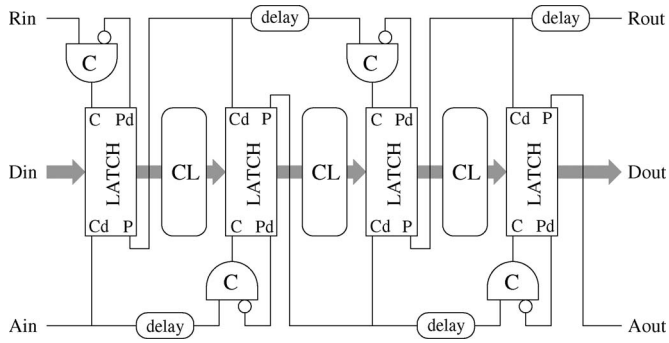

Fig. 12. Sutherland's micropipeline.

C. Two-Phase Signaling

In two-phase protocols, all events are signaling, and there is no semantic difference between the rising and falling events of a signal (see Fig. 8).

The two-phase signaling protocol was initially proposed by Seitz [100] and later used by Sutherland in the *Micropipelines* [111], shown in Fig. 12. The synchronization layer is based on Muller's pipeline. However, the two-phase interpretation of the events requires a special type of latch called a capture–pass (CP) latch, shown in Fig. 13.

The events on the `C` and `P` signals alternate to let the latch *capture* and *pass* data alternatively. The signals `Cd` and `Pd` are delayed versions of `C` and `P`, indicating that the capture and pass operations have completed, respectively. An alternative scheme was also proposed by substituting the CP latches by double edge-triggered latches [123].

A debate in the asynchronous community is whether two-phase protocols are superior to four-phase ones. On the one hand, two-phase control might be less power consuming and

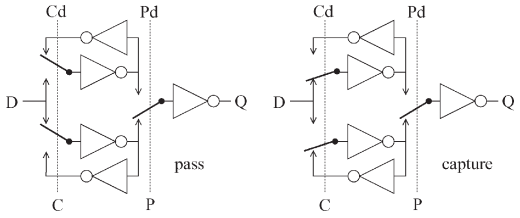

Fig. 13. CP latch.

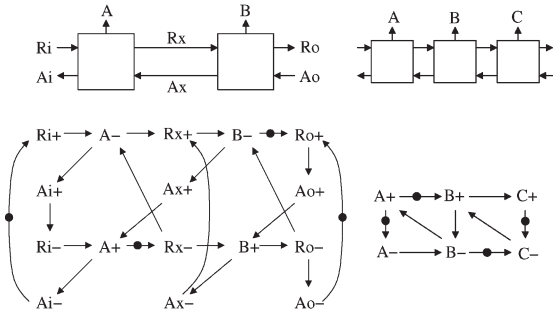

Fig. 14. Semidecoupled four-phase controller.

higher throughput, since it avoids the return-to-zero events. However, in logic terms, two-phase controllers are often more complex than four phase. An example is the special latches required for micropipelines. In addition, the return-to-zero phase can overlap with the next signaling phase to increases the throughput of a four-phase protocol. Moreover, power consumption depends on many factors: the capacitive loads of storage, the power consumed by the C-elements versus the simpler gates used in four-phase protocol, the length of the wires, and other factors.

The convenience or inconvenience of each protocol may depend on the particular requirements of each application: area, speed, and power.

D. Large Variety of Protocols for Latch Controllers

The previous sections have presented some of the earliest handshake protocols for the synchronization of asynchronous circuits. However, many different schemes have been studied and proposed.

A remarkable effort has been devoted to study four-phase signaling for latch controllers. The reason for that is the extensive variety of protocols that can be devised by exploring different ways of interleaving the return-to-zero events [13], [15], [39], [69], [75].

As an example, we next discuss the semidecoupled four-phase controller proposed in [39]. A behavioral specification is shown in Fig. 14. The $R_{i,x,o}$ and $A_{i,x,o}$ signals represent the req and ack signals of the controllers, whereas A, B, and C represent the enable signals of adjacent latches. The diagram at the left describes all the details of the protocol between two adjacent controllers. The diagram at the right describes the observable behavior when only considering the enable signals of the latches. Compared with Muller's pipeline, the semidecoupled protocol offers more concurrence for the return-to-zero events (e.g., as shown in Fig. 14, events $A-$ and $B+$ can happen concurrently). This reduces the length of the potentially critical handshake cycle between R_i and A_i, thus reducing the latency of the controller.

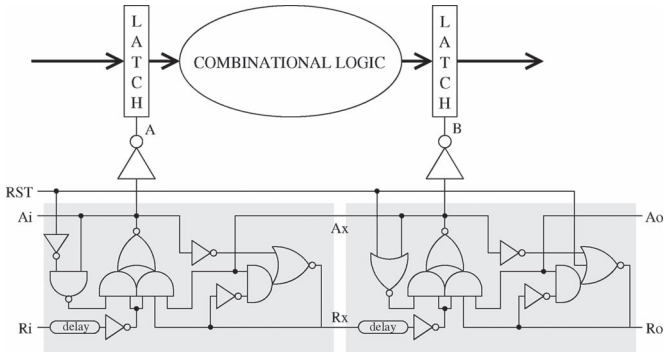

Fig. 15. Implementation of the semidecoupled four-phase controller.

An implementation of the semidecoupled controller is shown in Fig. 15. The two latches play the role of master and slave (or *vice versa*) in the original synchronous design. Therefore, the initialization of the controllers corresponding to two adjacent latches has to be different: Latch A is assumed to contain valid information, whereas latch B is assumed to be "void." Thus, the first enable pulse after resetting is produced in latch B, capturing the data stored in latch A.

Other more sophisticated schemes have also been proposed, like the ones using single-track protocols [11], [110]. Both schemes implement two-phase handshake mechanisms that only use one wire to implement the req/ack signaling. The schemes are based on the fact that the pull-up and pull-down networks of the wire are distributed between the sender and the receiver.

Singh and Nowick, in MOUSETRAP [104], proposed a latch-based high-speed pipeline in which the synchronization logic is implemented with simply one XOR gate for each latch. The protocol requires specific timing assumptions for correct operation.

Finally, it is worth mentioning the work done on dynamic pipelines [118], proposing high-speed schemes that exploit the concurrence between the precharge and evaluate phases of dynamic logic at different pipeline stages. Lines *et al.* [66], [72], [90] presented an approach for quasi-DI scheme for fine-grain pipelines. Recently, more advanced approaches with enhanced concurrence have achieved important improvements in throughput [73], [102], [103]. In conclusion, a variety of handshake controllers provides the designer with a variety of implementations to obtain an optimal handshake protocol for their particular application.

E. Join and Fork Blocks

To implement arbitrary elastic netlists using the handshake scheme shown in Fig. 7, it is necessary to provide an implementation for the Join and Fork blocks.

We next explain a very simple approach that can be used for any two- or four-phase protocol implemented with a pair of req/ack signals. We use the example in Fig. 16 that shows a latch-based circuit in which the shadow boxes represent latches. The letters inside the latches indicate their polarity: L stands for "active low" (i.e., the latch is open when the clock has zero value and opaque, otherwise) and H stands for "active high" (the latch is open when the clock is high). Note that this example does not explicitly distinguish between master

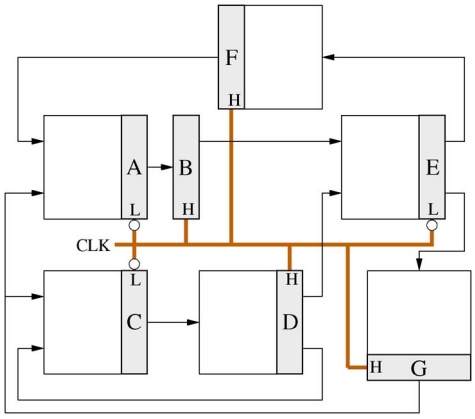

Fig. 16. Latch-based synchronous circuit.

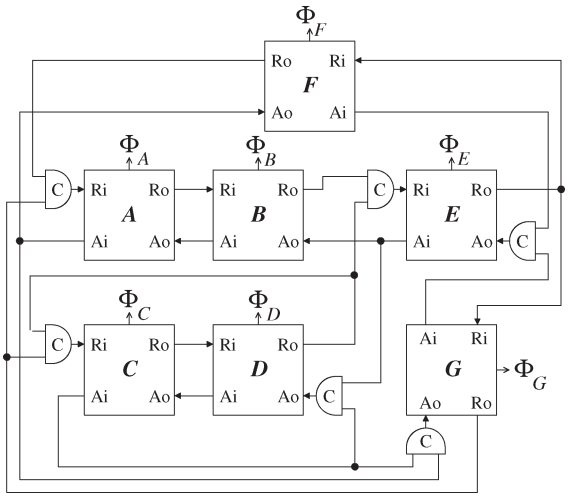

Fig. 17. Asynchronous control for the circuit in Fig. 16.

and slave latches and contemplates the possibility of having combinational logic between any pair of latches.

The synchronization of components with multiple inputs and outputs is shown in Fig. 17. The boxes represent the handshake controllers that generate the enable signals Φ_i for each latch. The Join block is implemented as the conjunction (C-element) of the corresponding req signals, whereas the Fork block is implemented as the conjunction of the ack signals. Some extensions for supporting Join and Fork blocks in dynamic pipelines have also been proposed [91].

Informally, the conjunction of these events can be interpreted as follows:

> "A block can perform an operation when all its inputs are available (C-element at Ri) and all its outputs have been consumed (*C-element at Ao*)."

With this scheme, the control for any arbitrary netlist can be built using C-elements and one-input/one-output latch controllers.

V. Synchronous Elasticity

This section describes the details of specific implementations of synchronous elasticity.

The original work by Carloni *et al.* [22] on latency-insensitive design described a handshaking protocol between two signals, *void* (not valid) and *stop*, interacting with a *relay station* implemented with two edge-triggered registers and a multiplexor, similar to the one shown in Fig. 5(b).

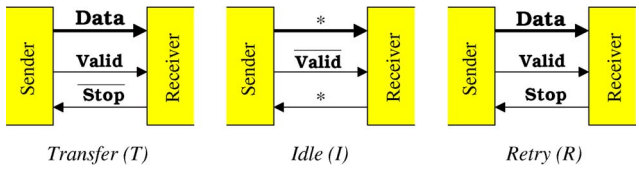

Fig. 18. SELF protocol.

The first latch-based implementation of synchronous elasticity was presented in [52]. Later, a similar implementation was used in [31] to propose an automatic procedure for elasticization of synchronous circuits. The formal specification and properties of the handshake protocol was later presented in [61]. Synchronous elasticity is sometimes referred to as pseudoasynchronous design [60], synchronous handshake circuits [92], and synchronous emulation of asynchronous circuits [89].

In this section, we will give more details about the design of controllers for latch-based implementations. There are two main reasons for this choice.

1) Latch-based implementations are more efficient in area, delay, and power.
2) The datapath can be identical to the one used for asynchronous elasticity described in Section IV.

As it will be shown later in this section, synchronous elasticity is achieved by augmenting a synchronous circuit with a clock-gating distributed controller that implements a synchronous handshake. The area overhead of this controller is minimal, particularly with respect to synchronous circuits that already have clock-gating control. The elastic controller typically introduces no delay overhead, since the computation and propagation delays of the control signals are shorter than delays of the corresponding computation blocks. The elastic controllers drive the enabling signals on the local clock lines. The depth of the enabling logic is similar to what is used in standard clock gating, and therefore, the contribution to *jitter* is expected to be similar to a standard synchronous design with clock gating.

A. Synchronous Elastic Protocol

To discuss the characteristics of synchronous elastic protocols, we will focus on the one presented in [31] and [61] [synchronous elastic flow (SELF)]. This protocol does not differ significantly from the one presented in [52], and it is conceptually similar to the one presented in [22], even though the latter is oriented to flip-flop-based implementations and requires some extra signals for the multiplexor of the EBs.

The two handshake signals, valid (V) and stop (S), determine three possible states in an elastic channel (see Fig. 18).

- $(\mathbf{T})$ **Transfer**, $(V \wedge \neg S)$: The sender provides valid data, and the receiver accepts it.
- $(\mathbf{I})$ **Idle**, $(\neg V)$: The sender does not provide valid data.
- $(\mathbf{R})$ **Retry**, $(V \wedge S)$: The sender provides valid data but the receiver does not accept it.

The sender has a *persistent* behavior when a *Retry* cycle is produced: It maintains the valid data until the receiver is able to read it. The language observed at the elastic channel can be described by the regular expression: $(I^*R^*T)^*$, indicating that every transfer may be preceded by a sequence of *Idle*

TABLE I
TRACE COMMITTING THE SELF PROTOCOL

Cycle	0	1	2	3	4	5	6	7	8	9
Data	$*$	A	B	B	B	C	$*$	$*$	D	D
Valid	0	1	1	1	1	1	0	0	1	1
Stop	0	0	1	1	0	0	0	1	1	0
State	I	T	R	R	T	T	I	I	R	T

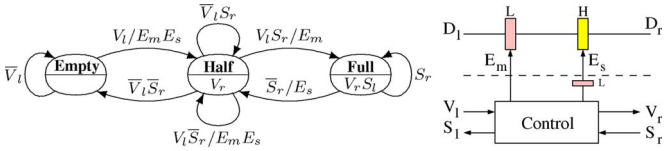

Fig. 19. Control specification for the latch-based EB.

cycles followed by a sequence of *Retry* cycles. The absence of a subtrace RI implies the persistence of the behavior. Table I shows an example of a trace transmitting the values $A-D$. When $V=0$, the value at the data bus is irrelevant (cycles 0, 6, and 7). The receiver can issue a *Stop* even when the sender does not send valid data (cycle 7). In the cycle following *Retry*, the sender persistently maintains the same valid data as in the previous cycle during cycles 3, 4, and 9.

B. Latch-Based Elastic Controllers

Fig. 19 shows the finite state machine (FSM) specifications for the control of a latch-based EB and the overall structure of the design. The transparent latches are shown with single boxes, labeled with the phase of the clock. The control drives latches with enable signals. To simplify the drawing, the clock lines are not shown. An enable signal for transparent latches must be emitted on the opposite phase and be stable during the active phase of the latch. Thus, the E_s signal for the slave latch is emitted on the L phase.

The FSM specification shown in Fig. 19 is similar to the specification of a two-slot FIFO. In the *Empty* state, no valid data are stored in the latches.

In the *Half-full* state, the slave latch stores one valid data item. Finally, in the *Full* state, both latches store valid data, and the EB emits a *stop* to the sender. The specification is a mixed Moore/Mealy FSM with some outputs associated with the states and some other with the transitions. For example, the transition from the *Half-full* state to the *Full* state occurs when the input channel carries valid information (V_l is high), but the output channel is blocked (S_r is high). An output signal enabling the master latch is emitted (E_m). In addition, the valid bit is emitted to the output channel $(V_r=1)$, since this Moore style signal is emitted at any transition from the *Half-full* state.

After encoding *Empty*, *Half-full*, and *Full* states of this FSM with $(V_r, S_l)=(0,0),(1,0),(1,1)$ correspondingly, we can derive the implementation shown in Fig. 20(a), where flip-flops are drawn as two back-to-back transparent latches. By splitting the flip-flops and retiming the latches, a fully symmetric latch-based implementation can be obtained [Fig. 20(b)].

C. Join and Fork

In general, EBs can have multiple input/output channels. This can be supported by using elastic Fork and Join control structures. Fig. 21(a) shows an implementation of a Join. The output valid signal is only asserted when both inputs are valid.

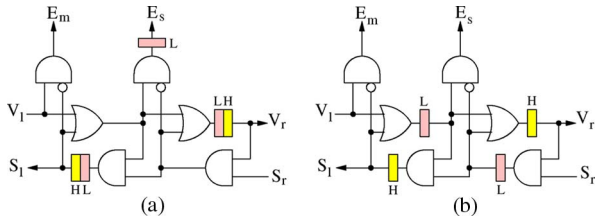

Fig. 20. Two implementations of an EB control.

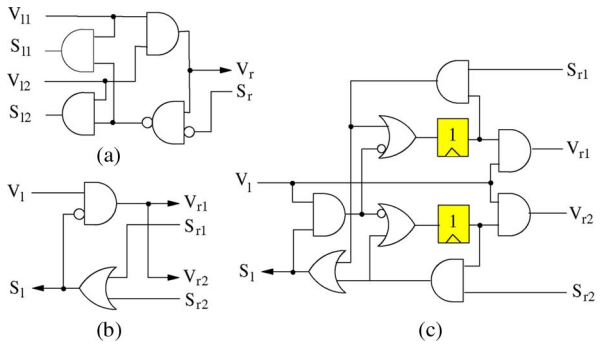

Fig. 21. Controllers for elastic Join and Forks. (a) Join. (b) Lazy fork. (c) Eager fork.

Otherwise, the incoming valid inputs are stopped. This construction allows the composing of multiple Joins together in a treelike structure.

Fig. 21(b) shows a Lazy Fork. The controller waits for both receivers to be ready $(S=0)$ before sending the data.[4] A more efficient structure shown in Fig. 21(c), the Eager Fork, can send data to each receiver independently as soon as it is ready to accept it. The two flip-flops are required to "remember" which output channels already received the data. This structure offers performance advantages when the two output channels have different back-pressures.

VI. EDA Synthesis Flow for Elastic Circuits

The approaches presented in the previous two sections to transform an inelastic circuit into an elastic one can be automated by a common EDA synthesis flow that is mostly independent from the fact that the final implementation can be either synchronous or asynchronous.

In this section, a synthesis flow for latch-based datapaths is presented, following the schemes described in [22], [31], and [52] for synchronous elasticity and in [16], [34], [59], and [113] for asynchronous elasticity.[5] A similar flow could be easily devised for alternative implementations of the EBs, such as the ones shown in Fig. 5(b) and (c).

A. Automatic Synthesis

The fundamental ingredients for the synthesis recipe are the control blocks that generate the enable signals for the latches and fork–join modules in the datapath.

1) The latch controllers that generate the enable signals for the latches. For example, the semidecoupled latch

[4]This implementation is identical to the one presented in [52].

[5]The automated flow for asynchronous elasticity has also been called *desynchronization* [34].

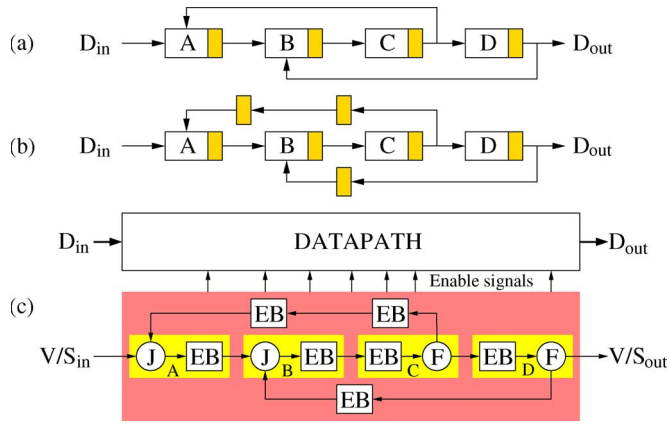

Fig. 22. Elasticization of a synchronous circuit.

controller could be used for asynchronous elasticity (see Fig. 15), or the EB controller in Fig. 20 for synchronous elasticity.

2) The Join and Fork blocks. For asynchronous elasticity, C-elements are sufficient to implement these blocks (as in Fig. 17) regardless the protocol used for the latch controllers. For synchronous elasticity, the designs shown in Fig. 21 can be used.

The steps for the transformation of an inelastic circuit [e.g., Fig. 6(a)] into an elastic circuit [e.g., Fig. 6(b) or (c)] are as follows.

1) Group the flip-flops into multibit registers. The registers determine the level of granularity at which elasticity is applied. Each register will be associated with one controller.
2) Remove the clock signal and replace each flip-flop with an EB [Fig. 5(d)].
3) Add the required *bubbles* to improve the performance of the circuit (see Section VIII-B).
4) Create the control layer that generates the enable signals for the latches. It has the following components:
   a) one latch controller for each EB;
   b) a Join block at the input of each latch controller receiving multiple inputs;
   c) a Fork block at the output of each latch controller sending multiple outputs.
5) In case of synchronous elasticity, connect the clock signal to the latch controllers and the join/fork blocks (if required). In case of asynchronous elasticity, add the matched delays for the appropriate timing among controllers. Timing will be discussed in Section VI-B.

An example showing the result of the transformations is presented next. Fig. 22(a) shows a synchronous datapath in which the shadowed boxes correspond to registers. Fig. 22(b) shows the same datapath after having added some bubbles (see Section VIII-B) for performance optimization. Fig. 22(c) shows the architecture of the control. The EB boxes represent the controllers for the latches. The `F` and `J` boxes represent the join and fork blocks, respectively. The arrows in the control represent pairs of handshake signals, with the arrow following the forward direction. The `V/S` signals represent the handshake signals that interact with the environment (valid/stop for synchronous or req/ack for asynchronous).

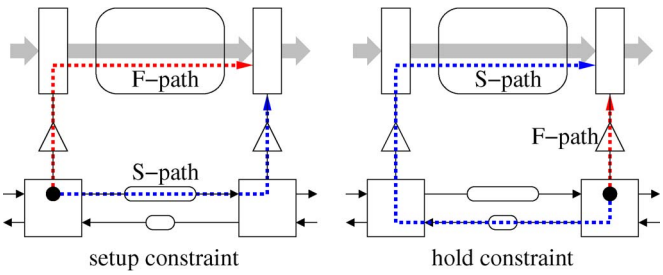

Fig. 23. Setup and hold constraints for asynchronous circuits.

B. Timing

This section briefly discusses some timing issues related to the previous synthesis flow.

In case of synchronous elastic circuits, a conventional two-phase clocking scheme for latch-based circuits can be used. When the logic between pairs of latches is unbalanced, *time borrowing* is typically used to compensate for the difference of delays between adjacent stages [27]. This situation occurs when the flip-flops are split into pairs of master/slave latches, without any logic between master and slave. Most timing-analysis tools provide support for latch-based designs and time borrowing.

Timing is less conventional when dealing with asynchronous circuits. However, the underlying constraints for the correctness of the circuit are not much different from the ones used in synchronous circuits: *setup* and *hold* constraints.

The timing constraints for asynchronous circuits are slightly different than those for the synchronous case, but they are still based on *setup* and *hold* constraints. Fig. 23 shows the timing paths for the data transfer between two latches in an asynchronous elastic circuit. In both cases, the arrival time of two paths, one fast (`F-path`) and another slow (`S-path`), are competing. Both paths have a common anchor point, represented as a thick dot. This point must be identified inside the latch controller and depends on its specific implementation. The timing constraints have the form

$$\delta_{\min}(\texttt{S-path}) > \delta_{\max}(\texttt{F-path}) \tag{1}$$

where $\delta_{\min}$ and $\delta_{\max}$ represent the minimum and maximum delays of the path, respectively. These constraints are the ones that determine the value of the matched delays. Timing constraints for asynchronous blocks from the design library can be generated using formal verification techniques. These constraints can be later validated using standard STA tools and used in standard synthesis flows [108].

The setup constraint, also called *long-path constraint*, guarantees that the enabling pulse of the receiving latch arrives after the data arrive at the input of the latch. In case of time borrowing, this constraint must refer to the falling edge of the pulse. The `F-path` covers the path from the launching controller to the input of the receiving latch. The `S-path` covers the path from the launching controller to the enable signal of the receiving latch through the matched delay of the req signals.

The hold constraint, also called *short-path constraint*, prevents data overwriting. The `F-path` covers the path from the receiving controller to the enable signal of the latch, whereas the `S-path` covers the path from the receiving controller back to the input of the latch through the ack signal and the launching controller and latch.

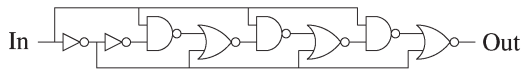

Fig. 24. Asymmetric delay for four-phase signaling.

For four-phase signaling, the previous constraints may involve "two rounds" across the same logic, for the active phase and the return-to-zero. In this case, the matched delays are crossed twice for the same constraint. For efficiency reasons, asymmetric delays are often used, thus minimizing the return-to-zero delay. Fig. 24 shows a possible implementation of an asymmetric delay with a long rising delay and a short falling delay.

VII. Completion Detection

Synchronous circuits are typically based on the assumption that a computation within every pipeline stage must be always completed within one clock cycle. Moreover, during the design process, the clock-cycle period is assumed to have a fixed duration. Most synchronization mechanisms are based on a feedforward clock network from the oscillator. All delay uncertainties in the clock tree, the combinational logic, and the setup-hold constraints of the sequential logic must be taken care of by means of appropriate worst-case margins. This style may not be appropriate for designs where some uncertainty can result in very large margins. For example, Hanson *et al.* [48] report that the clock period of a circuit in 65-nm technology must be increased by 230% for an ultralow power design with $\mathrm{Vdd} = 300$ mV (aggressively reduced voltage) to eliminate late-mode errors introduced by variability.

Modern on-die active control techniques (e.g., [42], [84]) allow switching the design into different power-performance states with different clock frequencies (and clock switched off in deep-sleep states). However, within each mode of operation, the clock cycle (and, hence, the deadline for every computation) is fixed. Time borrowing and clock scheduling that are applied in advanced synchronous circuits redistribute delay slack among different stages of computation and can reduce worst-case margins for individual stages and, as a result, the overall waste for delay margins.

There have also been recent attempts to reduce delay penalties in synchronous design due to manufacturing-process uncertainties by using STA [14], which is used to model the impact of correlated and independent variability sources on performance.[6]

However, the earlier techniques cannot adjust the duration of the clock period dynamically according to the data patterns that occur during the execution. This can be done by applying *completion detection* for signaling completion of operations instead of estimating their worst-case delays during the design process. We now discuss how completion design can be done in asynchronous and synchronous elastic circuits.

A. Asynchronous Circuits With Matched Delays

Most asynchronous design styles use some timing assumptions to correctly coordinate and synchronize computations.

[6]A discussion on the delay penalties in synchronous systems related to manufacturing-process uncertainties and the asynchronous alternatives can be found in [5] and [112].

The synchronization scheme for the asynchronous design shown in Fig. 6(c) is based on inserting matched delays into the request and acknowledgment paths of the controller. This design style is often called *bundled data*. It assumes that the maximum delay of each combinational logic island is smaller than that of a reference logic path, which is called a matched delay [105]. The durations of matched delays in different stages need not be the same. The simplest way of selecting the value for the delays is by applying standard STA to the corresponding computation block, assuming the worst-case delays and then adding some safety margin.

The design of the matched delays must be done taking into account the potential sources of variability, either static (process variations) or dynamic (voltage, temperature, noise). For this reason, a multicorner analysis with on-chip variability parameters must be performed. This would mainly affect the delay constraints shown in (1). For each constraint and variability parameter, the minimum and maximum delays must be considered for the left- and right-hand sides of the inequality, respectively.

Matched delays are implemented using the same gate library (typically as a chain of inverters) and are subject to the same operating conditions (temperature, voltage) *if placed in the same region as the corresponding blocks of the datapath*. This may result in consistent tracking of the datapath delays by the matched delays and allows to reduce the design margins.

The margins associated with the matched delays depend on technological parameters and the expected performance and yield. Conservative margins contribute to improve yield, since they provide more tolerance to variability. However, they have a negative impact on performance. On the other hand, small margins can result in better performance at the expense of sacrificing yield.

To adjust the value of the postmanufacturing delays according to the manufacturing and operating conditions (temperature, voltage) or to the type of data that are currently being processed, it is possible to introduce a completion detector based on multiplexing of *different* delay chains with different values of delays matched against particular cases of execution or manufacturing [41], [83].

A simple example of adjustable matched delay studied in the literature is with application to the variable-delay adders. Most additions do not require a complete carry propagation from the least to the most significant bit. By including extra logic, it is possible to monitor the carry propagation and deliver a completion signal sooner [40], [87].

B. Asynchronous Elastic Circuits With DI Codes

Fig. 25(a) shows an alternative scheme for designing asynchronous circuits. The datapath in this paper uses DI codes [116]. For such codes, it is always possible to detect when the operation is completed. Data are latched by stage $i+1$ when it detects the arrival of a new value from stage i. Because this completion information is encoded with the data, the circuit automatically adapts to timing variations arising from changes in the operating conditions or sensitivity to data patterns.

A simple DI code that allows for an easy completion detection is *a dual-rail code* in which every bit is encoded with two wires (perhaps the first use of this code with completion detection is published in [80] and [101]).

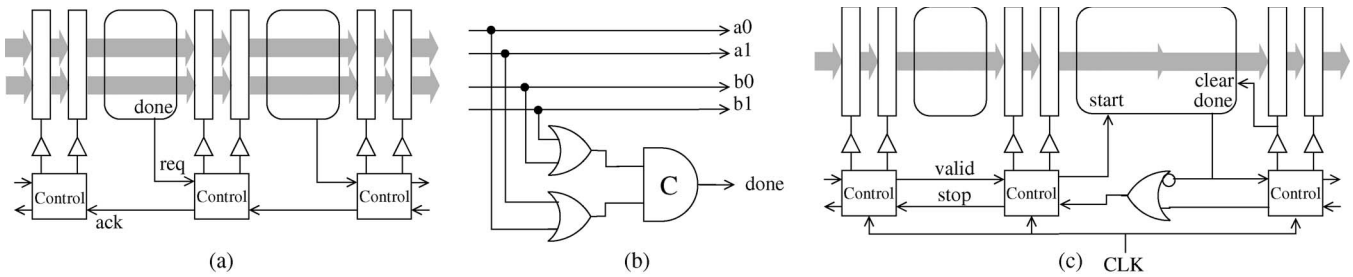

Fig. 25. (a) Asynchronous elastic design using DI code in the datapath and completion detection instead of matching delays. (b) Example of completion detection for QDI dual-rail code. (c) Synchronous elastic design with variable-latency unit and completion detection.

A dual-rail protocol uses two wires per bit, e.g., (a0, a1), for a single bit a in Fig. 25(b). Values of (a0, a1) equal to (1, 0) and (1, 0) represent "*valid data*" (logic 0 and 1, respectively). Code (0, 0) represents "no data" (a so-called *spacer*). The code (1, 1) is not used.[7] Every transition between two sequential valid data transfers should move through a spacer code. This allows for the implementing of a four-phase handshake protocol as follows.

1) The sender issues a valid code (1, 0) or (0, 1).
2) The receiver consumes the code after its validity is indicated by the completion detector and sets the acknowledgment high.
3) The sender responds by issuing the spacer code (0, 0).
4) The receiver acknowledges the spacer after it is indicated by the same completion detector by setting the acknowledgment low.

On an n-bit channel, a new codeword is received when exactly one wire in each of the n wire pairs makes a transition. The spacer is received when all pairs make a transition to the spacer. An implementation of such code recognizers is shown in Fig. 25(b): The OR gates distinguish valid data from spacer at each wire pair, whereas the n-input C-element changes the $done$ signal when all pairs are valid $(done = 1)$ or all pairs are spacers $(done = 0)$. Data are latched by the next stage after the completion signal $done$ is set [106].

While completion detection allows the measurement of real delays, it also requires overhead for completion detection and for the use of DI codes (such as dual rail). Hence, the actual advantages with respect to other design styles strongly depend on the class of applications.

C. Synchronous Elastic Circuits With Variable-Latency Units

Fig. 25(c) shows an extension of the basic synchronous elastic circuit from Fig. 6(b) to accommodate variable latency of operation in the second stage of the design. The *valid* signal of the stage starts the computation. The input latch is stalled until the $done$ signal reports the completion of the operation (which may occur in a few clock cycles). This triggers latching the result by the output latch and, if necessary, clears small internal FSMs inside the variable-latency unit.

Synchronous elastic pipelining presents a convenient framework for inserting variable-latency blocks, since the pipeline can tolerate variability in latency of operations. The variability in computation delays in this design style must be counted in multiples of cycles. Implementing variable-latency blocks requires synchronous completion detectors.

A typical approach for designing variable-latency units is called *telescopic units* [7], [8], [109].

An alternative approach is based on the partitioning of the original computational unit into several subunits to form an iterative network [117]. Each subunit is designed to distinguishing data words from transient words using some specific encoding, thus producing a completion signal. A typical case is a ripple carry adder with a scheme that detects the completion of the carry propagation. The adder is subdivided into k partial adders, and the variable latency of the adder is translated into a variable number of clock cycles, between 1 and k, that are required to complete the operation [19].

VIII. Performance Analysis and Optimization of Elastic Circuits

A. Performance Analysis

Performance analysis has received extensive attention in the last decades. The existing techniques can be classified into three groups: simulation-based [20], [76], [86], [121], partial-order [25], [50], and Markov analysis methods [62], [120]. In addition, fast hierarchical methods can be used [43]. To cover all of these techniques would require a tutorial on its own, so in this paper, we will pick a simple technique based on graph theory that is both simple and sufficient to illustrate the concepts of this section.

Let us start by analyzing the performance of a simple elastic circuit (a ring), consisting of six stages, as shown in Fig. 26(a). The formal model used in this paper to analyze an elastic circuit is a timed MG (TMG) [81], where transitions correspond to the combinational logic in stages, pairs of complementary arcs denote latches, and the marking represents the distribution of the data. Transitions are annotated with the delay of the combinational logic: It can be a natural number if the system is synchronous (number of clock cycles) or a real number if the system is asynchronous (e.g., delay in picoseconds). In the asynchronous setting, more detailed models can be defined that capture particular types of delays for a stage of the circuit [56]: function evaluation and reset delay, completion detection delay, and control overhead delays for evaluation and reset. For the sake of clarity, we summarize all these delays into only one number. This simplification can be dropped if a richer model is used. For instance, the full buffer channel net [6] annotates delays in the arcs instead of the transitions of the Petri net, thus allowing forward and backward latencies to be distinguished.

[7]Obviously, different encoding can be used, e.g., a spacer can be coded with (1, 1) or sometimes as (0, 0) and sometimes as (1, 1).

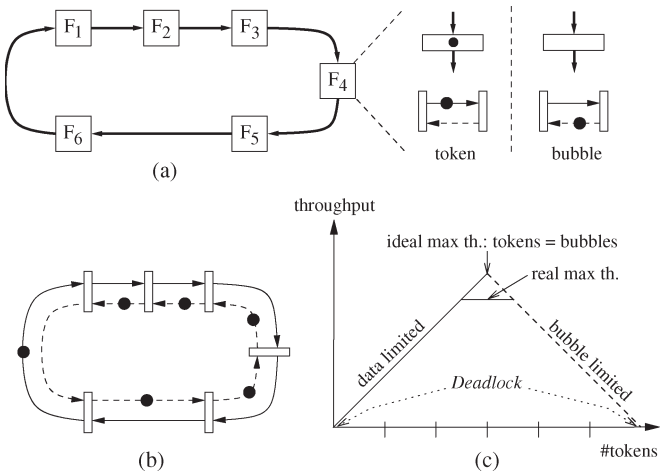

Fig. 26. (a) Elastic circuit. (b) Corresponding MG. (c) Throughput versus number of tokens.

In general, these delays are unequal with forward latencies typically being greater.

Tokens and bubbles (see Section III-C) are represented by the marking of complementary arcs, as shown in the right part of Fig. 26(a). In particular, a token is represented by a marker on the forward solid arrow, whereas a bubble is represented by a marker on the backward dashed arrow. Fig. 26(b) shows the TMG corresponding to the ring shown in Fig. 26(a), with only one datum in stage F_1. In this example, we assume unitary delays on all transitions and therefore do not explicitly annotate delays on transitions. Additionally, notice that (for convenience only) we draw the flow of data with solid lines (*forward arcs*), whereas the flow of back-pressure is drawn with dashed lines (*backward arcs*).

The performance of an elastic circuit is usually represented by its *throughput*, defined as the number of valid tokens that flow through a stage per time unit. Assuming unit delay in all stages shown in Fig. 26(a) and given that there is only one token in the circuit, its throughput is 1/6. In the TMG, this can be observed by the forward cycle with one token and six transitions. A question arises: what is the throughput with respect to its token distribution? Section VIII-B will introduce strategies for improving the performance of an arbitrary elastic circuit. In this paper, we analyze the performance of this toy ring with respect to the tokens and bubbles it has. Fig. 26(c) shows an interesting principle [45], [119]: by adding tokens into the ring, the throughput can increase up to a maximal point, in which the number of tokens is equal to the number of bubbles for identical forward and backward latencies.[8] As indicated in the figure, in reality, the maximal throughput achievable might be less than the ideal throughput, due to slow stages that limit the performance of the whole system. The left part of the maximal point is known as *data limited*, because only the absence of tokens limits the performance improvement, whereas the right part is known as *bubble limited*, since the absence of space in the elastic circuit limits the flow of tokens, thus degrading the performance. Deadlock points are also shown in the diagram, corresponding to an elastic circuit with no data (left) or with no bubbles (right).

[8]In general, the optimal number of tokens and bubbles depends on the values of the forward and backward latencies in the ring [45], [66], [118].

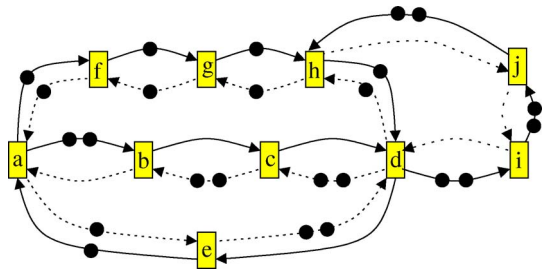

Fig. 27. Example of a TMG.

In general, an elastic circuit can be modeled with a TMG containing several cycles that are synchronized at particular points.

A TMG is a tuple $G = (T, A, M_0, \delta)$, where T is a set of transitions, $A \subseteq T \times T$ is a set of directed arcs, and $M_0 : A \rightarrow \mathbb{N}$ is a marking that assigns an initial number of tokens to each arc. Function $\delta : T \rightarrow \mathbb{R}^+ \cup \{0\}$ assigns a nonnegative delay to every transition. In a TMG, once a transition t is enabled, it fires after $\delta(t)$ time units. An example of a TMG is shown in Fig. 27. This TMG models latches of capacity two, and all transitions have unitary delay.

The throughput analysis of an arbitrary elastic circuit modeled as a TMG must consider several rings (cycles in the TMG), synchronized at particular points, as it happens in the example shown in Fig. 27. In [94], two important facts are observed for a TMG.

1) The performance is defined by the throughput of its most stringent simple cycle.
2) All transitions have the same throughput.

Formally, if C is the set of simple directed cycles in a TMG, its throughput can be determined as [94]

$$\Theta = \min_{\mathbf{c} \in C} \frac{M_0(\mathbf{c})}{\sum_{t \in \mathbf{c}} \delta(t)} \tag{2}$$

where $M_0(\mathbf{c})$ denotes the sum of tokens in cycle $\mathbf{c}$. In the example shown in Fig. 27, the throughput is 1/4 (backward cycle $\{d, h, j, i\}$). Many efficient polynomial algorithms for computing the throughput of a TMG exist that do not require an exhaustive enumeration of all cycles [35], [53].

The previous technique can be easily adapted to handle different forward and backward latencies, as it is shown, for instance, in [6]: The idea is to annotate the delays in the arcs, instead of the transitions. Hence, forward latencies will be assigned to forward arcs in the TMG model, whereas backward arcs will encode the corresponding backward latencies. Equation (2) can be easily adapted to define the throughput in this extended model.

B. Performance Optimization by Slack Matching

Two techniques to improve the performance of elastic circuits are presented in this section.

Unbalanced fork–join structures in an elastic circuit cause tokens in different branches to arrive at a join stage at different times, making the earlier tokens stall until tokens from the slow branches arrive. This will cause further tokens in the backward direction of the stalled tokens to stall, degrading the performance of the circuit. Let us illustrate this phenomenon on the example shown in Fig. 28. In Fig. 28(a), a synchronous

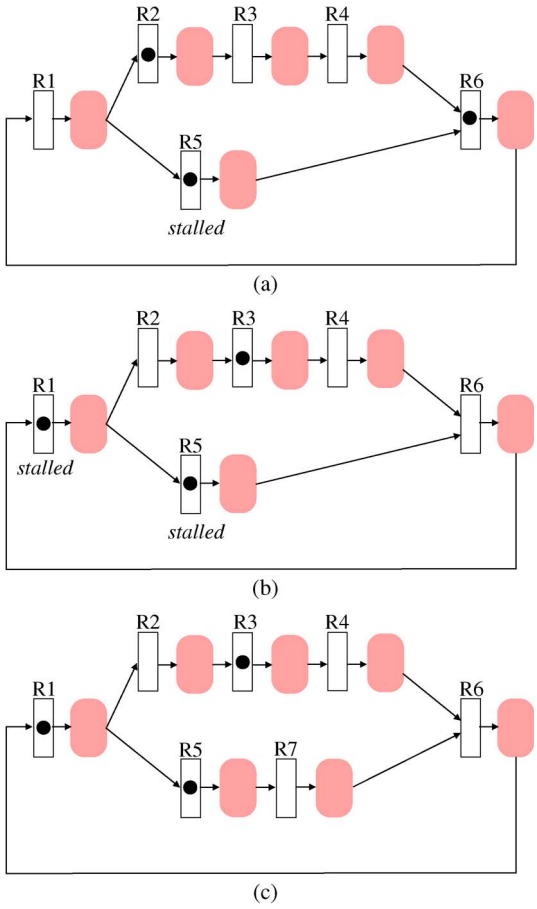

Fig. 28. (a) and (b) Effect of unbalanced branches. (c) Recycling fast branch for slack matching.

elastic circuit with two branches is shown. Assuming unit delays, the theoretical throughput is represented by the top cycle, with ratio 2/5 (two dots, five registers). Notice that, in the bottom cycle, there is one datum stalled in the initial state. This will cause, in the next clock tick, stalling the fork channel, as shown in Fig. 28(b). The fork channel is stalled one clock cycle until the slow branch can provide a token to the join channel. In summary, the stalling of the bottom (fast) branch produces the stalling of some channel in the slow branch. Hence, the throughput of the system is degraded to 2/6. This is because the bottom cycle is bubble limited.

One way to prevent this throughput degradation is by forcing tokens to arrive at join stages at the same time. Slack matching enforces this by increasing the capacity of fast branches with the aim of balancing fork–join branches [6], [70], [119]. In Fig. 28(c), an empty buffer has been inserted in the bottom branch to avoid the propagation of the stall to the fork channel. Even though the two branches are not totally balanced, this insertion is already sufficient to make the bottom cycle token limited, thus allowing the pipeline to reach the maximum throughput of 2/5.

Two techniques can be used to increase the capacity in short branches: The one applied in Fig. 28(c) is called *recycling* and is based on the insertion of empty buffers. A drawback of this transformation is that it might increase the cycle time of the system when these empty buffers create new critical cycles. The second transformation, known as *buffer sizing*, increments

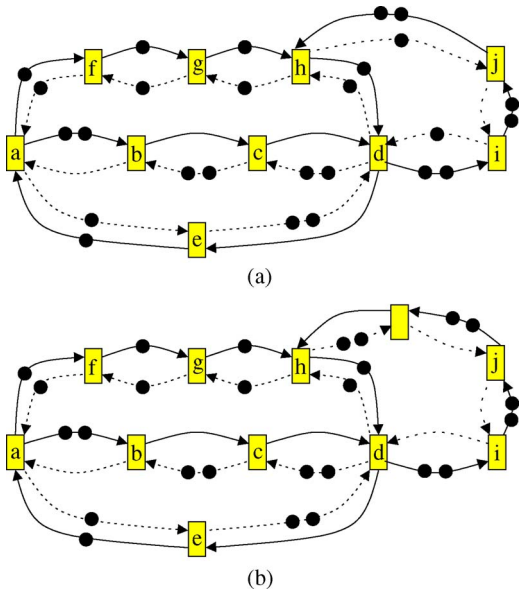

Fig. 29. Two techniques for slack matching for the TMG of Fig. 27. (a) Buffer sizing. (b) Recycling.

the capacity of buffers in short branches. The cost of buffer sizing is the need for extra logic to control the additional capacity of increased buffers [26], [67]. On the other hand, the cycle time is never degraded by buffer sizing. In general, the techniques for slack matching are usually based on the solution of a linear system of constraints. In this section, we will use the TMG model to illustrate how efficient algorithms can be derived for slack matching.

Buffer sizing: increasing buffer capacity. The modeling of the capacity increase of a buffer in an elastic circuit is done in the TMG by adding tokens in the corresponding backward arc. When the most stringent cycle of a TMG contains backward arcs, then increasing the capacity of the corresponding buffers may increase the throughput [67]. For instance, in Fig. 27, if the capacity of any of the channels in the cycle $\{d, h, j, i\}$ is augmented in one unit, then the throughput will rise to 2/4. With another increase (in one unit) of the capacity of any of these buffers, the cycle $\{d, h, j, i\}$ will have ratio 3/4. The resulting TMG is shown in Fig. 29(a). Note that, after these two transformations, the cycle $\{a, b, c, d, e\}$ becomes the most stringent cycle, determining the system throughput to be 3/5. Hence, from the TMG in Fig. 27, increasing buffer capacities in cycle $\{d, h, j, i\}$ in $2, 3, \ldots, k$ units will lead to the same throughput improvement. Unfortunately, the problem of determining the minimal buffer increase to reach the maximal throughput is NP-complete [96].

Recycling: adding bubbles. An optimization technique similar to buffer sizing is recycling [23], which also aims at increasing the ratio of critical cycles containing backward arcs. In synchronous circuits, recycling can be used to attain a given clock period by breaking long combinatorial paths: In this sense, it is a similar technique to *buffer insertion* in physical synthesis. An example of recycling applied to the TMG shown in Fig. 27 is shown in Fig. 29(b), with a bubble inserted between transitions h and j. This transformation achieves the same throughput improvement of buffer sizing on the same example. Recycling cannot always attain the same throughput

improvement as buffer sizing, because a zero-marked forward arc is inserted that may degrade the ratio of the cycle involved.

C. Early Evaluation

Conventional elastic systems rely on late evaluation: The computation is initiated only when all input data are available. This requirement can be too strict. Consider a multiplexor with the following behavior:

$$z = \texttt{if } s \texttt{ then } a \texttt{ else } b.$$

The result of a multiplexer can be produced by an *early evaluation* of the expression. For instance, if s and a are available and the value of s is *true*, then there is no need to wait for b to arrive before evaluating the output. In that case, the result $z = a$ can be produced right away. However, the value of b should be discarded when it arrives at the multiplexer without producing a new result at the output of the multiplexer.

In early evaluation, care must be taken in preventing the spurious enabling of functional units when the *nonrequired* inputs arrive later than the completion of the computation. A possible technique is the use of *negative tokens*, also called *antitokens*. Each time an early evaluation occurs, an antitoken is generated at every nonrequired input in such a way that when it meets the positive token, they annihilate. The antitokens can be *passive*, waiting for the positive token to arrive, or *active*, traveling in the backward direction to meet the positive token. The passive protocol is simpler to implement, but the active protocol may be advantageous in terms of power and performance.

Active antitokens implement a counterflow of information in the elastic circuits and therefore are reminiscent of the counterflow pipeline from [107]. The idea of passive antitokens for early evaluation was used in [57] and [122], extending Petri nets for handling OR causality and nodes with arbitrary guard functions.

To incorporate early evaluation in a elastic circuit, the Join module of a synchronous or asynchronous elastic controller must be modified: The AND evaluation function [shown in Fig. 21(a)] is replaced with a corresponding early evaluation guard function. The guard function can be data dependent and listens to a few inputs from the datapath. For the multiplexor example, the early evaluation guard needs to have a select signal of the multiplexor to make correct decisions on when antitokens can be issued and in which channel. In addition, some hardware should be added to discard tokens of unneeded information, e.g., by generating and counting antitokens for a passive protocol or generating and propagating antitokens for an active protocol.

For synchronous elasticity, several approaches have been proposed for early evaluation using different schemes to account for the negative tokens in the flow [24], [29], [65]. In the EDA flow proposed in this paper, the implementation of early evaluation can be reduced to an enhancement of the Join controllers.

The strategy for incorporating active antitokens in [29] was based on a dual counterflow of tokens. In that case, the two flows were implemented using symmetric versions of the latch, join, and fork controllers. The protocol required four handshake signals: two for tokens and two for antitokens. In that protocol, passive antitokens were simply implemented as a simplification of the scheme for active antitokens. The details of the implementation can be found in [30], in which the formal verification of the controllers is also addressed.

Early evaluation has also been used in asynchronous controllers. In [95], the inputs of blocks with early evaluation are partitioned into early and late. Early evaluation is allowed when all early inputs have arrived. The block waits for all inputs to arrive before advancing to the next evaluation. A 35% improvement was reported in the performance of a MIPS microprocessor.

Active antitokens have also been proposed by [1] and [17] for the design of faster asynchronous pipelines. In this case, special care must be taken to avoid metastability conditions when tokens and antitokens cross each other. In [1], an elegant solution was proposed based on implementing a two-phase protocol in which the arrival of tokens (antitokens) and the acknowledgment of antitokens (tokens) is treated symmetrically. With such property, the protocol can be implemented with a metastability-free circuit without requiring the use of arbiters.

IX. Examples and Discussion

The widespread use of cache memories, the difficulty of having an accurate estimation of the worst-case execution time for software, and the use of multitasking kernels make many systems work in scenarios with high variability in which average-case performance is the main metric to estimate efficiency.

Along the same vein, design schemes with elastic components (either synchronous or asynchronous) that provide average-case delays may be acceptable for many applications, ranging from general-purpose computing to multimedia and wireless communications. If the average throughput of an elastic device is significantly higher than that of a traditionally designed device, then the performance advantage may be sufficient to accept moderately disruptive changes in the design flow.

Instead of minimizing the worst-case parameters, several approaches [4], [19], [37], [55], [60], [99] suggest optimizing circuits for typical modes of operation, i.e., to make circuits elastic. One of the most known examples of such approach (sometimes called *Better Than Worst-Case Design* [4]) is Razor [37]. It is based on the observation that many signals often stabilize before their worst-case delay in most operating conditions. Thus, these signals can be sampled using shorter clock periods, at the expense of sporadically violating the long-path (setup) constraint.

The Razor CPU is designed with double slave latches and an XOR gate in each master–slave pair, thus doubling the area overhead of each converted latch. The second slave is clocked later than the first slave, guaranteeing a safe operation. If the comparator detects a difference between the values sampled by each slave, it means that the first slave memorized an incorrect value. When such anomaly occurs, the processor "skips a beat" and restarts the pipeline with the value of the second latch, which is always correct. This mechanism could be considered as a scheme for completion detection in synchronous designs.

This approach may be appealing for low-power processors, but it also has an inherent problem that makes it unsuitable for some application-specific integrated-circuit (ASIC) designs.

Due to near-critical clocking, it is always possible for the first latch to become metastable [58]. To avoid circuit malfunction in Razor, a synchronizer is added to the output of the latch in such a way that the error is detected after a certain latency.[9] When an error is detected, the pipeline is flushed and restarted in a way similar to what is done for precise interrupts. However, it is very difficult, if not impossible, to use similar mechanisms for conventional ASICs.

Asynchronous implementations with completion detection, as demonstrated, e.g., in [54], [74], and [85], achieve similar goals with much simpler logic, since the delay of the logic is directly used to generate the synchronization signals in a feedback control fashion.

A version of the DLX microprocessor [49] was automatically synthesized using the desynchronization approach presented in Section IV [2], [34]. When comparing the average-case asynchronous with the worst-case synchronous performance, the desynchronized version was shown to be in average 10% faster in 90% of the cases [2]. Desynchronized mode of operation also demonstrated good tolerance to variability of design parameters [33]. DLX was also chosen as an example of efficiency (it combines the modularity of asynchronous design with the ease of synchronous approach) for both the latency-insensitive approach [21] and the synchronous elastic pipelining [31] presented in Section V.

The correct-by-construction nature of synchronous elastic pipelining [21], [61] enables its applicability at the latest stages of the design, when delays of data transfers have been calculated, without any impact on the functionality of the systems.

Synchronous elastic pipelining can be effectively combined with variable-latency blocks when the synthesis of these blocks can be supported by EDA tools. A well-known example of this approach is telescopic units, which can be applied to relatively large designs achieving performance gains higher than 25% on average [109].

The syntax-directed translation from concurrent languages is another paradigm targeting asynchronous elasticity. The most prominent examples are the languages *Haste* [93] and *Balsa* [36], which are synthesized into a set of handshake components that implement the functionality of the language constructs [10]. Its main forte is the inheritance of all the advantages of asynchronous circuits by controlling the activation of both control and datapath units (and, hence, power and energy consumption) at a very fine level with direct and explicit HDL support. The best-known examples in this framework are the ARM996HS [12], a 32-b RISC processor core part of the ARM9E family targeted at low-power robust design, and the SmartMX microcontroller, used in 80% of the world's smart passports because of its low-power consumption [47]. This paradigm enables the conception of purely asynchronous systems from the early design stages, thus taking advantage of the full power of asynchrony in terms of computation and communication. Unfortunately, this approach, which is attractive from the theoretical point of view, requires reeducation of RTL designers and rewriting of existing specifications, which is not realistic for large and expensive designs (see [112] for details). In the context of compiling high-level specifications with choices into asynchronous elastic systems, it was shown that, for a system to be functionally correct after introducing buffers, it must meet the requirements for slack elasticity [70].

An extreme case of asynchronous elastic pipelining—gate-level pipelining, can combine high throughput with low voltage, given the robustness achieved by the insensitivity to variability provided by gate-level completion detection. Furthermore, this approach is suitable for applications related to hardware security [112], since the combination of asynchronous elasticity and balanced dynamic gates is essential for the design of devices that are highly resistant to side-channel attacks (dual-rail dynamic gates are easier to balance, completion detection makes early propagation attack problematic, glitch attacks are not possible, etc., see [63], [64] for details).

One of the main trends in EDA today is to increase the level of abstraction at which systems are specified, thus moving from RTL to higher level languages. Many researchers believe that most systems in the area of SoC design are going to use the so-called electronic-system-level (ESL) flows. At the very least, ESL flow is supposed to incorporate a front-end with higher level specifications (e.g., SystemC or ANSI-C) into synthesis flow. There are several interesting publications related to ESL flow for asynchronous (bundled delay) design [44], [114], [115]. This flow is actually closer to a software C-compiler than a hardware synthesis flow.

Nevertheless, elasticity can play a relevant role in this area, enabling correct-by-construction refinements of the high-level specifications. The tolerance to delay or latency variations, while preserving correctness, may become a valuable feature for the construction of complex systems composed of heterogeneous blocks that continuously change their nonfunctional properties (e.g., power and timing).

X. Conclusion

Elasticity is a concept that goes beyond the particular implementation aspects of timing. This paper has presented a unified view of elasticity in synchronous and asynchronous pipelines, showing that the underlying theory can often be shared and reused in different scenarios.

This paper has shown that a common EDA flow can be devised for elastic circuits in which the synchronous or asynchronous nature of the design only affects the clocking scheme synthesized by the flow.

The relevance of variability in the timing behavior of circuits will surely increase the presence of different forms of elasticity in the next few years.

[9] A typical synchronizer requires at least two flip-flops, thus adding a latency of at least one cycle.

References

[1] M. Ampalam and M. Singh, "Counterflow pipelining: Architectural support for preemption in asynchronous systems using anti-tokens," in *Proc. ICCAD*, 2006, pp. 611–618.

[2] N. Andrikos, L. Lavagno, D. Pandini, and C. P. Sotiriou, "A fully-automated desynchronization flow for synchronous circuits," in *Proc. 44th DAC*, 2007, pp. 982–985.

[3] *AMBA Specification (Rev 2.0)*, ARM Limited, U.K., 1999.

[4] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, "Opportunities and challenges for better than worst-case design," in *Proc. ASP-DAC*, 2005, pp. 2–7.

[5] P. Beerel, J. Cortadella, and A. Kondratyev, "Bridging the gap between asynchronous design and designers," in *Proc. VLSI Des. Conf.*, Mumbai, India, 2004, pp. 18–20. (tutorial).

[6] P. Beerel, A. Lines, M. Davies, and N.-H. Kim, "Slack matching asynchronous designs," in *Proc. 12th IEEE Int. Symp. Asynchronous Circuits Syst.*, Mar. 2006, pp. 184–194.

[7] L. Benini, E. Macii, M. Poncino, and G. De Micheli, "Telescopic units: A new paradigm for performance optimization of VLSI designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 3, pp. 220–232, Mar. 1998.

[8] L. Benini, G. De Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino, "Automatic synthesis of large telescopic units based on near-minimum timed supersetting," *IEEE Trans. Comput.*, vol. 48, no. 8, pp. 769–779, Aug. 1999.

[9] C. H. van Berkel, M. B. Josephs, and S. M. Nowick, "Scanning the technology: Applications of asynchronous circuits," *Proc. IEEE*, vol. 87, no. 2, pp. 223–233, Feb. 1999.

[10] K. van Berkel, *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*, vol. 5. Cambridge, U.K.: Cambridge Univ. Press, 1993.

[11] K. van Berkel and A. Bink, "Single-track handshaking signaling with application to micropipelines and handshake circuits," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar. 1996, pp. 122–133.

[12] A. Bink and R. York, "ARM996HS: The first licensable, clockless 32-bit processor core," *IEEE Micro*, vol. 27, no. 2, pp. 58–68, Mar./Apr. 2007.

[13] G. Birtwistle and K. Stevens, "The family of 4-phase latch protocols," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 2008, pp. 71–82.

[14] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, "Statistical timing analysis: From basic principles to state of the art," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 4, pp. 589–607, Apr. 2008.

[15] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou, "Handshake protocols for de-synchronization," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 2004, pp. 149–158.

[16] A. Branover, R. Kol, and R. Ginosar, "Asynchronous design by conversion: Converting synchronous circuits into asynchronous ones," in *Proc. DATE*, Feb. 2004, pp. 870–875.

[17] C. Brej, "Early output logic and anti-tokens," Ph.D. dissertation, Univ. Manchester, Manchester, U.K., 2005.

[18] J. A. Brzozowski and C.-J. H. Seger, *Asynchronous Circuits*. New York: Springer-Verlag, 1995.

[19] A. Burg, F. K. Gürkaynak, H. Kaeslin, and W. Fichtner, "Variable delay ripple carry adder with carry chain interrupt detection," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2003, pp. 113–116.

[20] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," Ph.D. dissertation, California Inst. Technol., Pasadena, CA, 1991.

[21] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1059–1076, Sep. 2001.

[22] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sagiovanni-Vincentelli, "A methodology for correct-by-construction latency insensitive design," in *Proc. ICCAD*, Nov. 1999, pp. 309–315.

[23] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Performance analysis and optimization of latency insensitive systems," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2000, pp. 361–367.

[24] M. Casu and L. Macchiarulo, "Adaptive latency-insensitive protocols," *IEEE Des. Test Comput.*, vol. 24, no. 5, pp. 442–452, Sep./Oct. 2007.

[25] S. Chakraborty, K. Yun, and D. Dill, "Timing analysis of asynchronous systems using time separation of events," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 8, pp. 1061–1076, Aug. 1999.

[26] T. Chelcea and S. Nowick, "Robust interfaces for mixed-timing systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 8, pp. 857–873, Aug. 2004.

[27] D. Chinnery, K. Keutzer, J. Sanghavi, E. Killian, and K. Sheth, "Automatic replacement of flip-flops by latches in ASICs," in *Closing the Gap Between ASIC & Custom*, D. Chinnery and K. Keutzer, Eds. Norwell, MA: Kluwer, 2002, ch. 7.

[28] T.-A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," Ph.D. dissertation, MIT Lab. Comput. Sci., Cambridge, MA, Jun. 1987.

[29] J. Cortadella and M. Kishinevsky, "Synchronous elastic circuits with early evaluation and token counterflow," in *Proc. ACM/IEEE DAC*, Jun. 2007, pp. 416–419.

[30] J. Cortadella and M. Kishinevskyin "Synchronous elastic circuits with early evaluation and token counterflow," Universitat Politècnica de Catalunya, Barcelona, Spain, Tech. Rep. LSI-07-13-R, 2007. [Online]. Available: www.lsi.upc.edu/~techreps/files/R07-13.zip

[31] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proc. ACM/IEEE DAC*, Jul. 2006, pp. 657–662.

[32] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces*. New York: Springer-Verlag, 2002.

[33] J. Cortadella, A. Kondratyev, L. Lavagno, and C. P. Sotiriou, "Coping with the variability of combinational logic delays," in *Proc. 22nd IEEE Int. Conf. Comput. Des.*, 2004, pp. 505–508.

[34] J. Cortadella, A. Kondratyev, L. Lavagno, and C. P. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 1904–1921, Oct. 2006.

[35] A. Dasdan, S. S. Irani, and R. K. Gupta, "Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems," in *Proc. 36th Des. Autom. Conf.*, 1999, pp. 37–42.

[36] D. Edwards and A. Bardsley, "Balsa: An asynchronous hardware synthesis language," *Comput. J.*, vol. 45, no. 1, pp. 12–18, Jan. 2002.

[37] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: Circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, Nov./Dec. 2004.

[38] R. M. Fuhrer and S. M. Nowick, *Sequential Optimization of Asynchronous and Synchronous Finite-State Machines: Algorithms and Tools*. Norwell, MA: Kluwer, 2001.

[39] S. B. Furber and P. Day, "Four-phase micropipeline latch control circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 2, pp. 247–253, Jun. 1996.

[40] J. D. Garside, "A CMOS VLSI implementation of an asynchronous ALU," in *Asynchronous Design Methodologies*, vol. A-28, S. Furber and M. Edwards, Eds. Amsterdam, The Netherlands: Elsevier, 1993, pp. 181–207.

[41] J. D. Garside, W. J. Bainbridge, A. Bardsley, D. A. Edwards, S. B. Furber, J. Liu, D. W. Lloyd, S. Mohammadi, J. S. Pepper, O. Petlin, S. Temple, and J. V. Woods, "AMULET3i—An asynchronous system-on-chip," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 2000, pp. 162–175.

[42] G. Gerosa, S. Curtis, M. D'Addeo, B. Jiang, B. Kuttanna, F. Merchant, B. Patel, M. Taufique, and H. Samarchi, "A sub-1 W to 2 W low-power IA processor for mobile internet devices and ultra-mobile PCs in 45 nm hi-k metal gate CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2008, pp. 256–611.

[43] G. Gill, V. Gupta, and M. Singh, "Performance estimation and slack matching for pipelined asynchronous architectures with choice," in *Proc. ICCAD*, Nov. 2008, pp. 449–456.

[44] G. Venkataramani and S. C. Goldstein, "Leveraging protocol knowledge in slack matching," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 2006, pp. 724–729.

[45] M. R. Greenstreet and K. Steiglitz, "Bubbles can make self-timed pipelines fast," *J. VLSI Signal Process.*, vol. 2, no. 3, pp. 139–148, Nov. 1990.

[46] P. L. Guernic, J.-P. Talpin, and J.-C. L. Lann, "Polychrony for system design," *J. Circuits Syst. Comput.*, vol. 12, no. 3, pp. 261–304, Apr. 2003.

[47] *Handshake solutions empowers 80% of epassports globally*, 2006. [Online]. Available: http://www.handshakesolutions.com/News/

[48] S. Hanson, B. Zhai, D. Blaauw, D. Sylvester, A. Bryant, and X. Wang, "Energy optimality and variability in subthreshold design," in *Proc. ISLPED*, 2006, pp. 363–365.

[49] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 1990.

[50] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, "An algorithm for exact bounds on the time separation of events in concurrent systems," *IEEE Trans. Comput.*, vol. 44, no. 11, pp. 1306–1317, Nov. 1995.

[51] *IEEE Standard for a Versatile Backplane Bus: VMEbus*, IEEE Std 1014-1987 (R2008), 1987.

[52] H. M. Jacobson, P. N. Kudva, P. Bose, P. W. Cook, S. E. Schuster, E. G. Mercer, and C. J. Myers, "Synchronous interlocked pipelines," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 2002, pp. 3–12.

[53] R. Karp, "A characterization of the minimum cycle mean in a digraph," *Discr. Math.*, vol. 23, no. 3, pp. 309–311, Sep. 1978.

[54] C. Kelly, V. Ekanayake, and R. Manohar, "SNAP: A sensor-network asynchronous processor," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, May 2003, pp. 24–33.

[55] E. Kim, D.-I. Lee, H. Saito, H. Nakamura, J.-G. Lee, and T. Nanya, "Performance optimization of synchronous control units for datapaths with variable delay arithmetic units," in *Proc. ASP-DAC*, 2003, pp. 816–819.

[56] S. Kim and P. A. Beerel, "Pipeline optimization for asynchronous circuits: Complexity analysis and an efficient optimal algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 3, pp. 389–402, Mar. 2006.

[57] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky, *Concurrent Hardware: The Theory and Practice of Self-Timed Design*, ser. Series in Parallel Computing. New York: Wiley, 1994.

[58] L. Kleeman and A. Cantoni, "Metastable behavior in digital systems," *IEEE Des. Test Comput.*, vol. 4, no. 6, pp. 4–19, Dec. 1987.

[59] R. Kol and R. Ginosar, "A doubly-latched asynchronous pipeline," in *Proc. ICCD*, Oct. 1996, pp. 706–711.

[60] Y. Kondo, N. Ikumi, K. Ueno, J. Mori, and M. Hirano, "An early-completion-detecting ALU for a 1 GHz 64 b datapath," in *Proc. 43rd IEEE ISSCC. Dig. Tech. Papers*, Feb. 6–8, 1997, pp. 418–419. 497.

[61] S. Krstic, J. Cortadella, M. Kishinevsky, and J. O'Leary, "Synchronous elastic networks," in *Proc. FMCAD*, 2006, pp. 19–30.

[62] P. Kudva, G. Gopalakrishnan, and E. Brunvand, "Performance analysis and optimization for asynchronous circuits," in *Proc. ICCD*, Oct. 1994, pp. 221–224.

[63] K. Kulikowski, A. Smirnov, and A. Taubin, "Automated design of cryptographic devices resistant to multiple side-channel attacks," in *Proc. CHES*, Yokohama, Japan, 2006, pp. 399–413.

[64] K. Kulikowski, V. Venkataraman, Z. Wang, A. Taubin, and M. Karpovsky, "Asynchronous balanced gates tolerant to interconnect variability," in *Proc. IEEE Int. Symp. Circuits Syst.*, Seattle, WA, 2008, pp. 3190–3193.

[65] C.-H. Li and L. Carloni, "Leveraging local intracore information to increase global performance in block-based design of systems-on-chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 2, pp. 165–178, Feb. 2009.

[66] A. Lines, "Pipelined asynchronous circuits," M.S. thesis, California Inst. Technol., Pasadena, CA, 1998. (CaltechCSTR:1998.cs-tr-95-21).

[67] R. Lu and C.-K. Koh, "Performance optimization of latency insensitive systems through buffer queue sizing of communication channels," in *Proc. ICCAD*, Nov. 2003, pp. 227–231.

[68] R. Lu and C.-K. Koh, "Performance analysis of latency insensitive systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 3, pp. 469–483, Mar. 2006.

[69] R. Manohar, "An analysis of reshuffled handshaking expansions," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar. 2001, pp. 96–105.

[70] R. Manohar and A. J. Martin, "Slack elasticity in concurrent computing," in *Proc. 4th Int. Conf. Math. Program Construction*, J. Jeuring, Ed, 1998, vol. 1422, pp. 272–285.

[71] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Advanced Research in VLSI*, W. J. Dally, Ed. Cambridge, MA: MIT Press, 1990, pp. 263–278.

[72] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Pénzes, R. Southworth, and U. Cummings, "The design of an asynchronous MIPS R3000 microprocessor," in *Proc. Adv. Res. VLSI*, Sep. 1997, pp. 164–181.

[73] A. J. Martin and M. Nystrom, "Asynchronous techniques for system-on-chip design," *Proc. IEEE*, vol. 94, no. 6, pp. 1089–1120, Jun. 2006.

[74] A. J. Martin, M. Nyström, K. Papadantonakis, P. I. Pénzes, P. Prakash, C. G. Wong, J. Chang, K. S. Ko, B. Lee, E. Ou, J. Pugh, E.-V. Talvala, J. T. Tong, and A. Tura, "The lutonium: A sub-nanojoule asynchronous 8051 microcontroller," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, May 2003, pp. 14–23.

[75] P. B. McGee and S. M. Nowick, "A lattice-based framework for the classification and design of asynchronous pipelines," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2005, pp. 491–496.

[76] E. G. Mercer and C. J. Myers, "Stochastic cycle period analysis in timed circuits," in *Proc. Int. Symp. Circuits Syst.*, 2000, pp. 172–175.

[77] R. Milner, *Communication and Concurrency*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[78] C. E. Molnar, T.-P. Fang, and F. U. Rosenberger, "Synthesis of delay-insensitive modules," in *Proc. Chapel Hill Conf. Very Large Scale Integr.*, H. Fuchs, Ed, 1985, pp. 67–86.

[79] D. E. Muller, "Asynchronous logics and application to information processing," in *Proc. Symp. Appl. Switching Theory Space Technol.*, 1962, pp. 289–297.

[80] D. E. Muller and W. S. Bartky, "A theory of asynchronous circuits," in *Proc. Int. Symp. Theory Switching*, Apr. 1959, pp. 204–243.

[81] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[82] C. Myers, *Asynchronous Circuit Design*. New York: Wiley, 2001.

[83] S. Naffziger, B. Stackhouse, T. Grutkowski, D. Josephson, J. Desai, E. Alon, and M. Horowitz, "The implementation of a 2-core, multi-threaded itanium family processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 197–209, Jan. 2006.

[84] *Powerwise adaptive voltage scaling*. [Online]. Available: http://www.national.com/analog/powerwise

[85] L. Necchi, L. Lavagno, D. Pandini, and L. Vanzago, "An ultra-low energy asynchronous processor for wireless sensor networks," in *Proc. 12th IEEE Int. Symp. Asynchronous Circuits Syst.*, Mar. 13–15, 2006, pp. 78–85.

[86] C. D. Nielsen and M. Kishinevsky, "Performance analysis based on timing simulation," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 1994, pp. 70–76.

[87] S. M. Nowick, K. Y. Yun, and P. A. Beerel, "Speculative completion for the design of high-performance asynchronous dynamic adders," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 1997, pp. 210–223.

[88] *Open Core Protocol Specification*, OCP Int. Partnership, Portland, OR, 2007. release 2.2.

[89] J. O'Leary and G. Brown, "Synchronous emulation of asynchronous circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 2, pp. 205–209, Feb. 1997.

[90] R. O. Ozdag and P. A. Beerel, "High-speed QDI asynchronous pipelines," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 2002, pp. 13–22.

[91] R. O. Ozdag, M. Singh, P. A. Beerel, and S. M. Nowick, "High-speed non-linear asynchronous pipelines," in *Proc. DATE*, Mar. 2002, pp. 1000–1007.

[92] A. Peeters and K. van Berkel, "Synchronous handshake circuits," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar. 2001, pp. 86–95.

[93] A. M. G. Peeters, "Implementation of handshake components," in *Proc. 25 Years Communicating Sequential Processes*, 2004, pp. 98–132.

[94] C. V. Ramamoorthy and G. S. Ho, "Performance evaluation of asynchronous concurrent systems using Petri nets," *IEEE Trans. Softw. Eng.*, vol. SE-6, no. 5, pp. 440–449, Sep. 1980.

[95] R. Reese, M. Thornton, C. Traver, and D. Hemmendinger, "Early evaluation for performance enhancement in phased logic," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 4, pp. 532–550, Apr. 2005.

[96] J. Rodriquez-Beltran and A. Ramirez-Trevino, "Minimum initial marking in timed marked graphs," in *Proc. IEEE Int. Conf. SMC*, Oct. 2000, vol. 4, pp. 3004–3008.

[97] L. Y. Rosenblum and A. V. Yakovlev, "Signal graphs: From self-timed to timed ones," in *Proc. Int. Workshop Timed Petri Nets*, Torino, Italy, Jul. 1985, pp. 199–207.

[98] S. S. Sapatnekar, "Static timing analysis," in *The CRC Handbook of EDA for IC Design*, L. Scheffer, L. Lavagno, and G. Martin, Eds. Boca Raton, FL: CRC Press, 2006, pp. 6-1–6-17.

[99] T. Sato and I. Arita, "Constructive timing violation for improving energy efficiency," in *Compilers and Operating Systems for Low Power*. Norwell, MA: Kluwer, 2003, pp. 137–153.

[100] C. L. Seitz, "System timing," in *Introduction to VLSI Systems*, C. A. Mead and L. A. Conway, Eds. Reading, MA: Addison-Wesley, 1980, ch. 7.

[101] J. C. Sims and H. J. Gray, "Design criteria for autosynchronous circuits," in *Proc. Eastern Joint Comput. Conf. (AFIPS)*, Dec. 1958, vol. 14, pp. 94–99.

[102] M. Singh and S. M. Nowick, "The design of high-performance dynamic asynchronous pipelines: High-capacity style," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 11, pp. 1270–1283, Nov. 2007.

[103] M. Singh and S. M. Nowick, "The design of high-performance dynamic asynchronous pipelines: Lookahead style," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 11, pp. 1256–1269, Nov. 2007.

[104] M. Singh and S. M. Nowick, "MOUSETRAP: High-speed transition-signaling asynchronous pipelines," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 6, pp. 684–698, Jun. 2007.

[105] *Principles of Asynchronous Circuit Design: A Systems Perspective*, J. Sparsø and S. Furber, Eds. Norwell, MA: Kluwer, 2001.

[106] J. Sparsø, J. Staunstrup, and M. Dantzer-Sørensen, "Design of delay insensitive circuits using multi-ring structures," in *Proc. EURO-DAC*, Hamburg, Germany, Sep. 1992, pp. 15–20.

[107] R. F. Sproull, I. E. Sutherland, and C. E. Molnar, "The counterflow pipeline processor architecture," *IEEE Des. Test Comput.*, vol. 11, no. 3, pp. 48–59, Fall 1994.

[108] K. S. Stevens, Y. Xu, and V. Vij, "Characterization of asynchronous templates for integration into clocked CAD flows," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, 2009, pp. 151–161.
[109] Y.-S. Su, D.-C. Wang, S.-C. Chang, and M. Marek-Sadowska, "An efficient mechanism for performance optimization of variable-latency designs," in *Proc. 44th Annu. DAC*, 2007, pp. 976–981.
[110] I. Sutherland and S. Fairbanks, "GasP: A minimal FIFO control," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar. 2001, pp. 46–53.
[111] I. E. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.
[112] A. Taubin, J. Cortadella, L. Lavagno, A. Kondratyev, and A. Peeters, "Design automation of real life asynchronous devices and systems," *Found. Trends Electron. Des. Autom.*, vol. 2, no. 1, pp. 1–133, Jan. 2007.
[113] V. Varshavsky and V. Marakhovsky, "GALA (Globally Asynchronous—Locally Arbitrary) design," in *Concurrency and Hardware Design*, vol. 2549, J. Cortadella, A. Yakovlev, and G. Rozenberg, Eds. New York: Springer-Verlag, 2002, pp. 61–107.
[114] G. Venkataramani, T. Bjerregaard, T. Chelcea, and S. C. Goldstein, "Hardware compilation of application-specific memory-access interconnect," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 5, pp. 756–771, May 2006.
[115] G. Venkataramani, M. Budiu, T. Chelcea, and S. Goldstein, "C to asynchronous dataflow circuits: An end-to-end tool flow," in *Proc. IWLS*, Temecula, CA, Jun. 2004, pp. 501–508.
[116] T. Verhoeff, "Delay-insensitive codes—An overview," *Distrib. Comput.*, vol. 3, no. 1, pp. 1–8, Mar. 1988.
[117] W. M. Waite, "The production of completion signals by asynchronous, iterative networks," *IEEE Trans. Comput.*, vol. C-13, no. 2, pp. 83–86, Apr. 1964.
[118] T. E. Williams, "Self-timed rings and their application to division," Ph.D. dissertation, Stanford Univ. Press, Stanford, CA, Jun. 1991.
[119] T. E. Williams, "Performance of iterative computation in self-timed rings," *J. VLSI Signal Process.*, vol. 7, no. 1/2, pp. 17–31, Feb. 1994.
[120] A. Xie and P. A. Beerel, "Symbolic techniques for performance analysis of timed systems based on average time separation of events," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Apr. 1997, pp. 64–75.
[121] A. Xie and P. A. Beerel, "Performance analysis of asynchronous circuits and systems using stochastic timed Petri nets," in *Hardware Design and Petri Nets*, A. Yakovlev, L. Gomes, and L. Lavagno, Eds. Norwell, MA: Kluwer, Mar. 2000, pp. 239–268.
[122] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny, "On the models for asynchronous circuit behaviour with OR causality," *Form. Methods Syst. Des.*, vol. 9, no. 3, pp. 189–233, Nov. 1996.
[123] K. Y. Yun, P. A. Beerel, and J. Arceo, "High-performance asynchronous pipeline circuits," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar. 1996, pp. 17–28.

Josep Carmona received the M.S. and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1999 and 2004, respectively.

He was a Visiting Scholar at the University of Leiden, Leiden, The Netherlands, in 2003. He is currently a Lecturer with the Department of Software, Universitat Politècnica de Catalunya. His research interests include formal methods and computer-aided design of very large scale integration systems with special emphasis on asynchronous circuits, concurrent systems, logic synthesis, and nanocomputing.

Dr. Carmona was the recipient of the Best Paper Award at the 9th International Symposium on Application of Concurrency to System Design in 2009.

Jordi Cortadella (M'88) received the M.S. and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya, Barcelona, Spain, in 1985 and 1987, respectively.

He was a Visiting Scholar at the University of California, Berkeley, in 1988. He is currently a Professor with the Department of Software, Universitat Politècnica de Catalunya. He is currently also Chief Scientist with Elastix Solutions S.L., Barcelona. His research interests include formal methods and computer-aided design of very large scale integration systems with special emphasis on asynchronous circuits, concurrent systems, and logic synthesis. He has coauthored numerous research papers and has been invited to present tutorials at various conferences.

Dr. Cortadella has served on the technical committees of several international conferences in the field of design automation and concurrent systems. He was the recipient of the Best Paper Awards at the International Symposium on Advanced Research in Asynchronous Circuits and Systems and at the Design Automation Conference, both in 2004, and the International Symposium on Application of Concurrency to System Design in 2009. He was also the recipient of a Distinction for the Promotion of the University Research from the Generalitat de Catalunya in 2003.

Mike Kishinevsky (SM'96) received the M.S. and Ph.D. degrees in computer science from the Electrotechnical University of St. Petersburg, St. Petersburg, Russia.

He was a Research Fellow with the Russian Academy of Sciences, Moscow, Russia; a Senior Researcher at a start-up in asynchronous design (TRASSA); a Visiting Associate Professor at the Technical University of Denmark, Lyngby, Denmark; and a Professor with the University of Aizu, Aizu-Wakamatsu, Japan. In 1998, he joined Intel Corporation, Hillsboro, OR, where he is currently leading a research group in front-end design with the Strategic CAD Laboratories. He coauthored three books in asynchronous design and has published over 70 journal and conference papers.

Dr. Kishinevsky has served on the technical program committee at several conferences and workshops. He was the recipient of the Semiconductor Research Corporation Outstanding Mentor Award in 2004, the Best Paper Awards at the Design Automation Conference in 2004 and at the International Symposium on Application of Concurrency to System Design in 2009.

Alexander Taubin (SM'96) received the M.Sc. and Ph.D. degrees in computer science and engineering from the Electrotechnical University of St. Petersburg, St. Petersburg, Russia.

From 1979 to 1989, he was a Research Fellow with the Computer Department, St. Petersburg Mathematical Economics Institute, Russian Academy of Sciences, Moscow, Russia. From 1988 to 1993, he was a Senior Researcher with the R&D Coop TRASSA. From 1991 to 1992, he was a Postdoctoral Researcher with the Department of Advanced Research, Institute of Microelectronics, Zelenograd, Russia. From 1993 to 1999, he was a Professor with the Department of Computer Hardware, University of Aizu, Aizu-Wakamatsu, Japan. In 1999, he joined as a Senior Scientist with Theseus Logic, Inc., Sunnyvale, CA. He is currently with the Electrical and Computer Engineering Department, Boston University, Boston, MA, where he joined as an Associate Professor in 2002. His current research interests include design and design automation of asynchronous pipelined systems and high-security, high-speed, and low-power devices and systems. He coauthored three books in asynchronous design and has published more than 60 journal and conference papers.

Dr. Taubin was the recipient of the Best Paper Award from the 11th Design, Automation and Test in Europe (DATE 2008) Conference. He has served on the technical committees of several international conferences in his field. He was Program Cochair of the 2nd and the 14th International Symposium on Advanced Research in Asynchronous Circuits and Systems (Async1996 and Async2008).